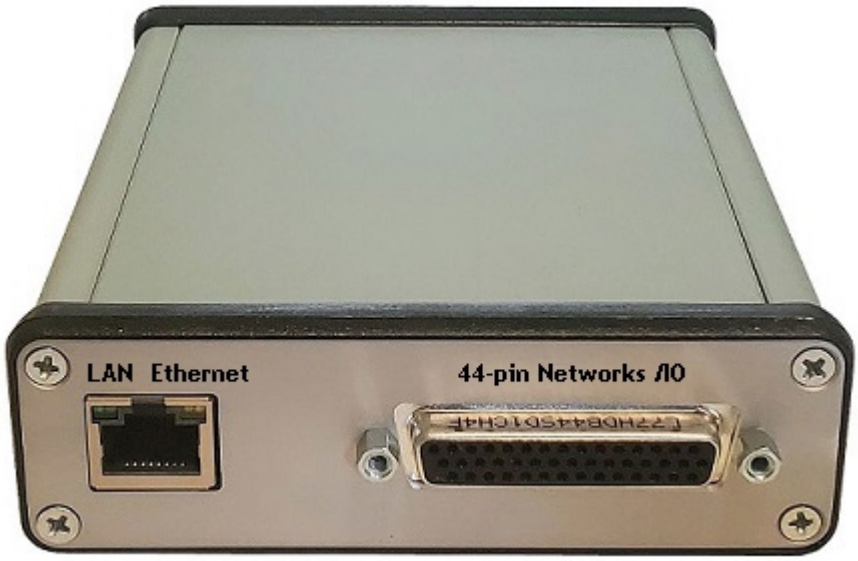
Introduction

This document is a user manual on how to operate and utilize Netway product.

Front Panel



Back Panel

Netway product is designed to monitor and control all multiplex networks utilized in automotive industry.
Netway device is a stand-alone hardware which can be connected to PC application via USB or Serial interface.
PXI/PCI card format is also available

List of multiplex networks / protocols include but not limited:

- CAN FD / CAN
- Flexray
- All **U**art **B**ased **P**rotocols (UBP): LIN, KWP2000, ISO-9141, J1708, etc.
- SPI/I2C/J1850-VPW/PWM
- Common Industry Protocols: J1939, J1708, J1587, J2617 (SENT/ SPC) and more
- Ethernet LAN

In addition to multiplex protocols, Netway device features up to 32 IO signals.
Each signal can processed as analog or digital, some of them as PWM.
Netway is a multi-channel device and can be configured to handle up to eight CAN FD, up to eight UBP, up to three Flexray and up to two SPI / I2C / J1850 independent channels.
Typical device configuration includes 2 CAN FD channels + 2 UBP channels + Ethernet LAN channel + 32 I/O signals.
When connected to PC, device can be configured with *Emulation* program which can be downloaded into device to enable it for real-time processing per emulation in stand alone mode.

Typical real-time tasks performed in stand alone mode:

- Message monitoring, analysis, processing, data logging, etc.
- Rest of vehicle node simulation
- Gateway between protocols
- End Of Line testing

Device details:

Power: 7.5-40V External power, otherwise works from USB power

Inputs: Analog/Digital/PWM

- Including 32 Digital signals ‘1’- above 3V, ‘0’-below 3V
- Including 32 Analog signals 0-16.8V range, 12 bit resolution
- Including 4 PWM input signals
- Including 4 input SENT/SPC (J2716) signals

Outputs: Digital/Analog/PWM

- Including 32 digital signals: Open drain, active low 50V/500mA
- Including 10 PWM outputs 0-65535 Hz/0-99% duty cycle
- Including 8 analog outputs 0-5V range
- Including 4 output SENT/SPC (J2716) signals

Other features:

- SD Card 8/16/32/64 GB card. File operations controlled by the emulation
- PC Interface Mini USB, a built-in serial bridge for firmware update
- Network Connector 44-pin HDB connector for power, networks, and I/O signals
- Dimensions 5.25" x 4.25" x 1.25" (aluminum enclosure)
- Netway desktop application. Compatible with Windows 7 and later (32 or 64 bit OS)
- CommonNetway.dll Type Library for custom control development (MS Studio 19)
- RTC on board, configurable power-down, power-up from CAN activities
- Stand-alone operations base on Emulation program created and downloaded by Netway application
- Product ID: NW92

Network protocols:

· CAN channels: Including two CAN FD channels (Cost $200)

- Additional **three** daughter boards with double CAN channels available (Cost $400 per DB)
- Transceiver: CAN FD high-speed

· UART channels two UART channels (Cost $400)

- Additional **three** daughter boards with double UART channels available (Cost $400 per DB)
- Transceivers LIN (KLINE) / RS485 (alternative RS232 option)

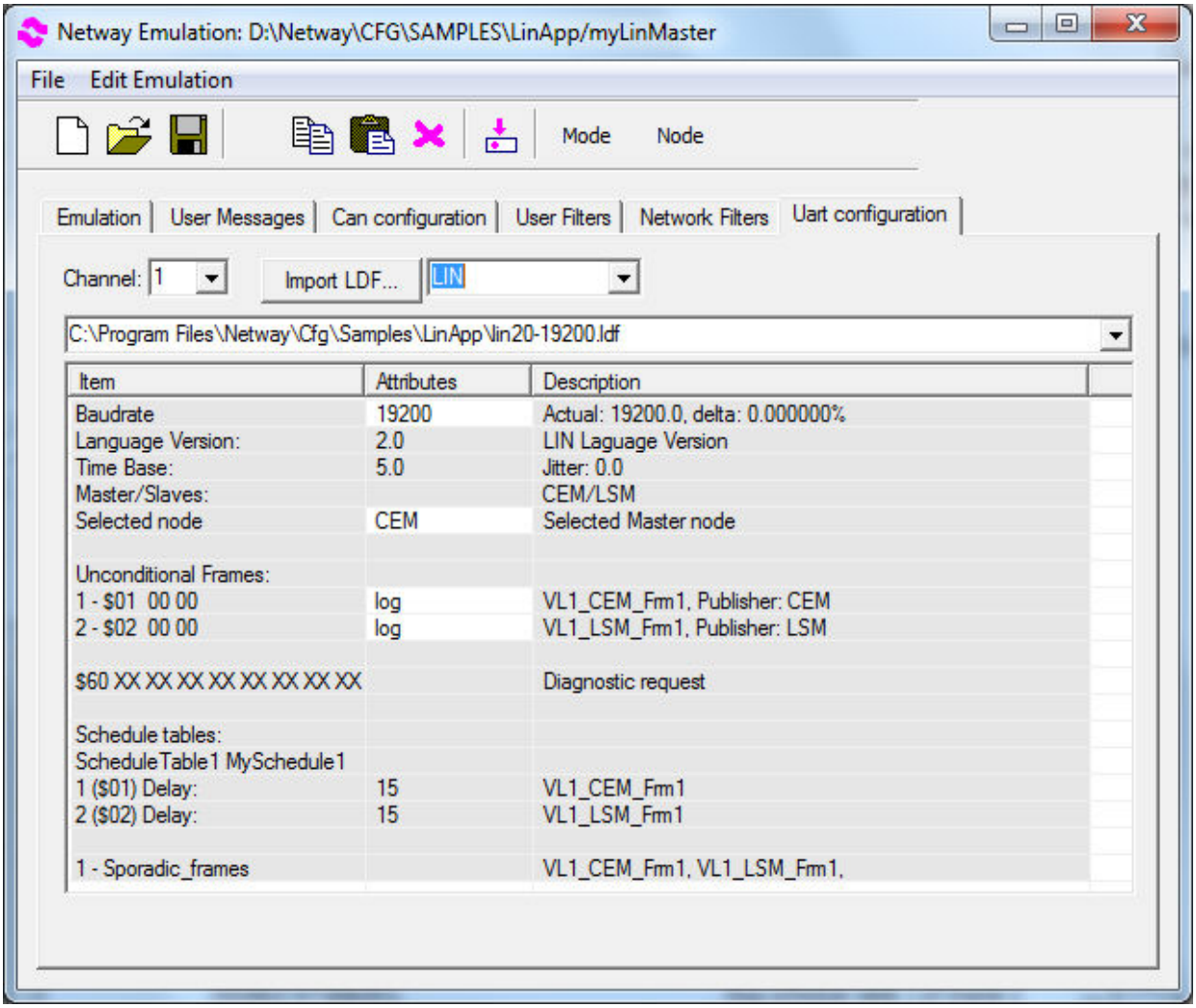
- Additional **two** SPI/IIC/J1850 daughter boards available (Cost $200 per DB)
- Additional **three** Flexray independent channel daughter boards available (Cost $400 per DB)
- LAN communication TCP/IP DHCP (Cost $200)

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

LIN

Note: This section assumes that a user is familiar with LIN protocol specifications (versions 1.0 - 2.1). Also note that LIN is a premium option and when available it is listed in About Netway Window: LIN protocol.
The LIN protocol can be utilized on the UART channels of a Netway device, The current Netway LIN implementation complies with LIN specification version 1.0 thru 2.1.
The Netway Software Package includes a demo Master/Slave LIN Application Emulation Programs in the **<…Netway\Cfg\Samples\LinApp>** subdirectory. This section uses a sample example as a starting point for learning how to use Netway devices with LIN protocol and to demonstrate a Netway device's LIN capabilities.
All the LIN network settings are contained in a LIN definition file (*.ldf) which is usually distributed to LIN module suppliers by vehicle manufacturers, however the **<LinApp>** subdirectory contains examples of such LDF files. Users can change the LIN parameters, frames, nodes, schedule table, signals, etc. in these files with any text editor (e.g. Notepad).

LIN channel 1 configuration:
Load LIN example emulation from C:\Program Files\Netway\Cfg\Samples\LinApp, select myLinMaster.emu file and download into the device. To see the LIN settings, switch to Uart configuration property page of Emulation dialog:

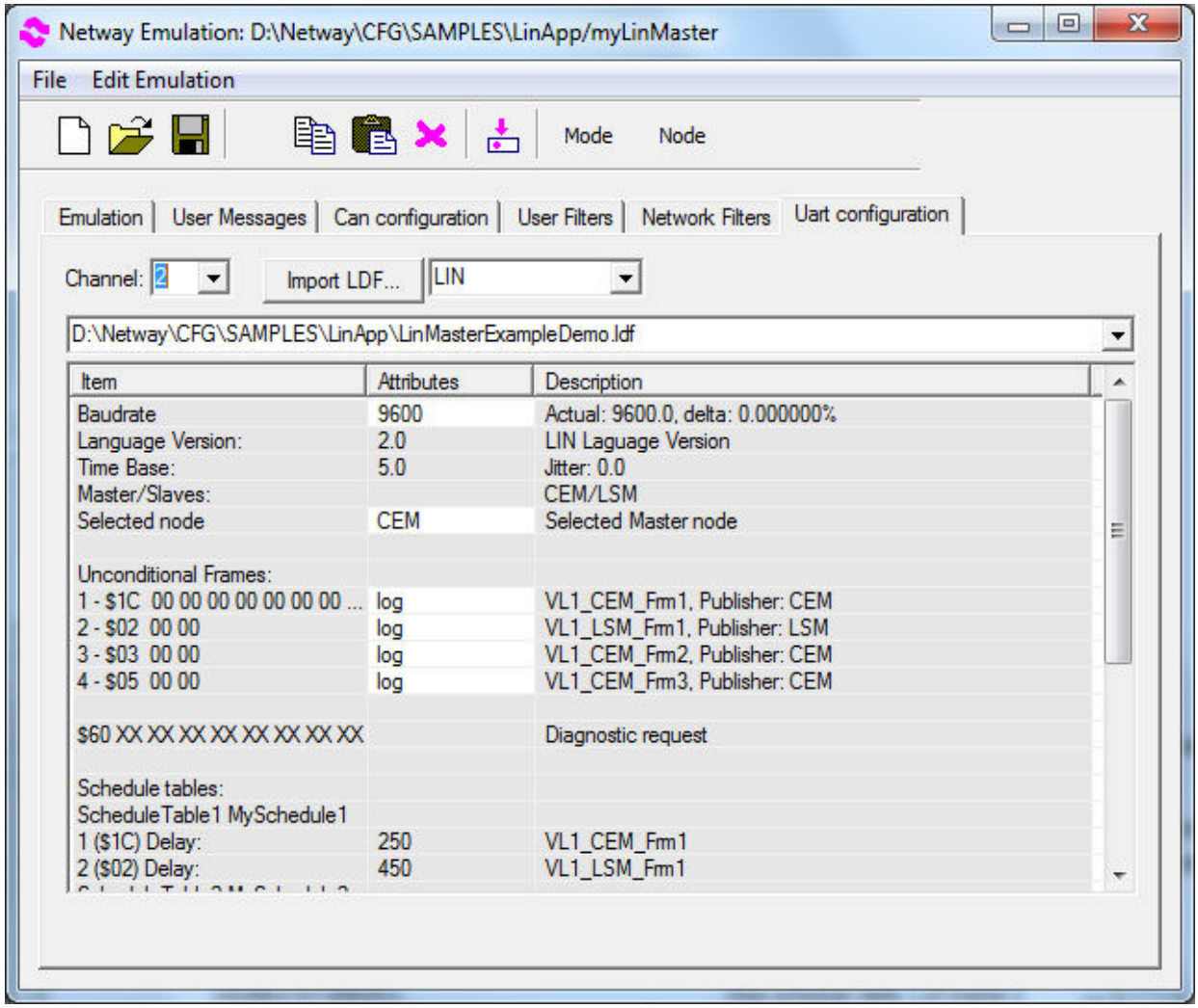


Note, the LIN is selected as UART protocol and parsed LDF is displayed in the settings window.

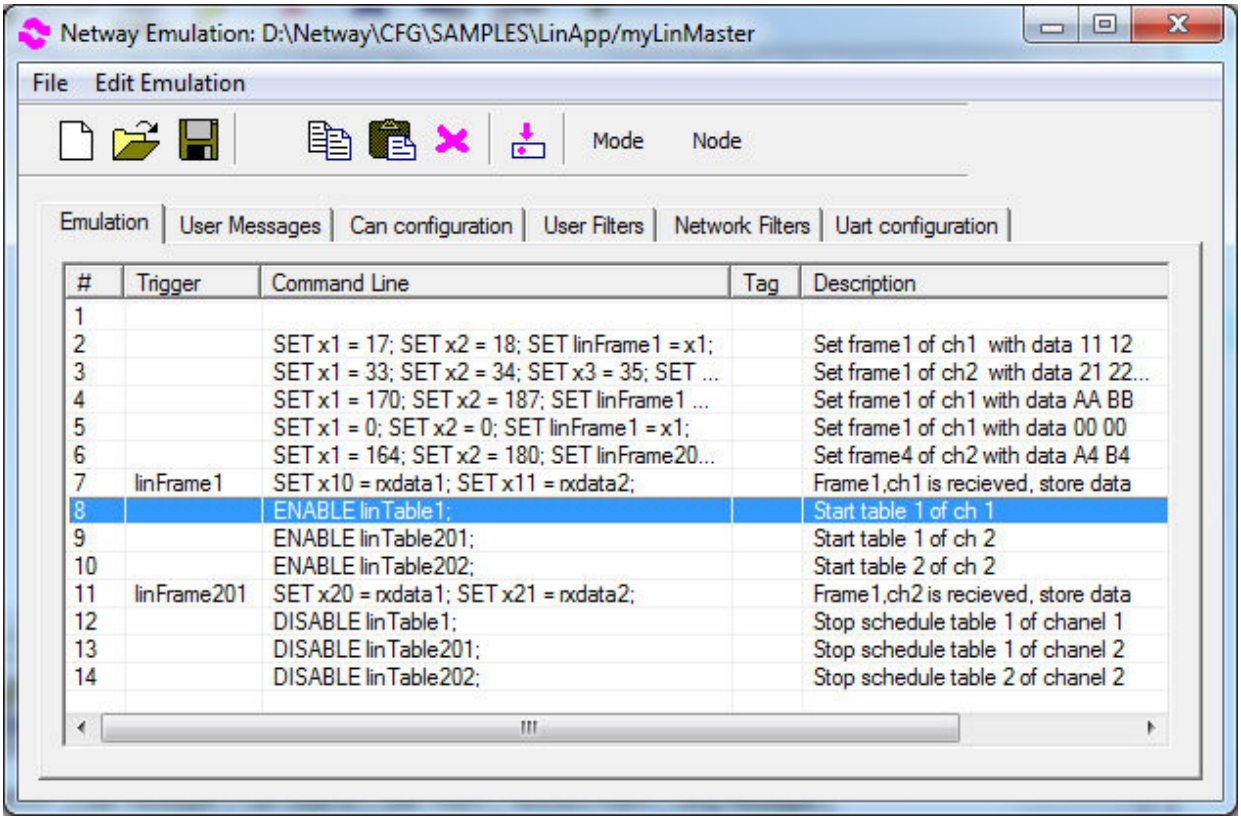
The 'white' fields are modifiable. User can modify baud rate, selected node and log/discard option for each unconditional frame. The remaining fields are greyed out and can be changed only by editing LDF in text editor.

LIN channel 2 configuration:

Use <Import LDF...> button to load a different LDF file or re-load modified LDF. Each LIN (UART) channel can be assigned to work with different LDF files, limited to maximum 4 (four) LDF files per emulation. In our example if we switch to channel 2 a different LDF file is displayed:



ENABLE / DISABLE linTableNNN commands:
Once emulation is loaded no schedule tables are active until the selected table is enabled.


Netway Emulation: D:\Netway\CFG\SAMPLES\LinApp/myLinMaster
File
Edit Emulation
Mode
Node
Emulation
User Messages
Can configuration
User Filters
Network Filters
Uart configuration
| Trigger | Command Line | Tag | Description
1
2 | SET x1 = 17; SET x2 = 18; SET linFrame1 = x1; | Set frame1 of ch1 with data 11 12
3 | SET x1 = 33; SET x2 = 34; SET x3 = 35; SET ... | Set frame1 of ch2 with data 21 22...
4 | SET x1 = 170; SET x2 = 187; SET linFrame1 ... | Set frame1 of ch1 with data AA BB
5 | SET x1 = 0; SET x2 = 0; SET linFrame1 = x1; | Set frame1 of ch1 with data 00 00
6 | SET x1 = 164; SET x2 = 180; SET linFrame20... | Set frame4 of ch2 with data A4 B4
7 | linFrame1 | SET x10 = rxdata1; SET x11 = rxdata2; | Frame1,ch1 is recieved, store data
8 | ENABLE linTable1; | Start table 1 of ch 1
9 | ENABLE linTable201; | Start table 1 of ch 2
10 | ENABLE linTable202; | Start table 2 of ch 2
11 | linFrame201 | SET x20 = rxdata1; SET x21 = rxdata2; | Frame1,ch2 is recieved, store data
12 | DISABLE linTable1; | Stop schedule table 1 of chanel 1
13 | DISABLE linTable201; | Stop schedule table 1 of chanel 2
14 | DISABLE linTable202; | Stop schedule table 2 of chanel 2

In this example Command lines 8, 9 and 10 are enabling schedule tables:
CL8: ENABLE linTable1; // enable schedule table 1 of channel 1
CL9: ENABLE linTable201; // enable schedule table 1 of channel 2
CL10: ENABLE linTable202; // enable schedule table 2 of channel 2

- **Note:** With ENABLE linTableXXX command currently active schedule table is disabled and XXX table becomes active (enabled)

The Command lines 12, 13 and 14 are used to disable a corresponding table if it is active:
CL12: DISABLE linTable1; // Stop schedule table 1 of channel 1
CL13: DISABLE linTable201; // Stop schedule table 1 of channel 2
CL14: DISABLE linTable202; // Stop schedule table 2 of channel 2

- **Note:** ENABLE/DIABLE linTableXXX commands take effect only if Master node is selected!

Setting LIN frames commands:
LIN frame data can be changed by the following commands:
CL2: SET x1 = 11h; SET x2 = 12h; SET linFrame1 = x1; //Set frame1 data of channel 1
CL3: SET x1 = 21h;SET x2 = 22h;...SET x8 = 28h;SET linFrame201 = x1; //Set frame1 data of channel 2

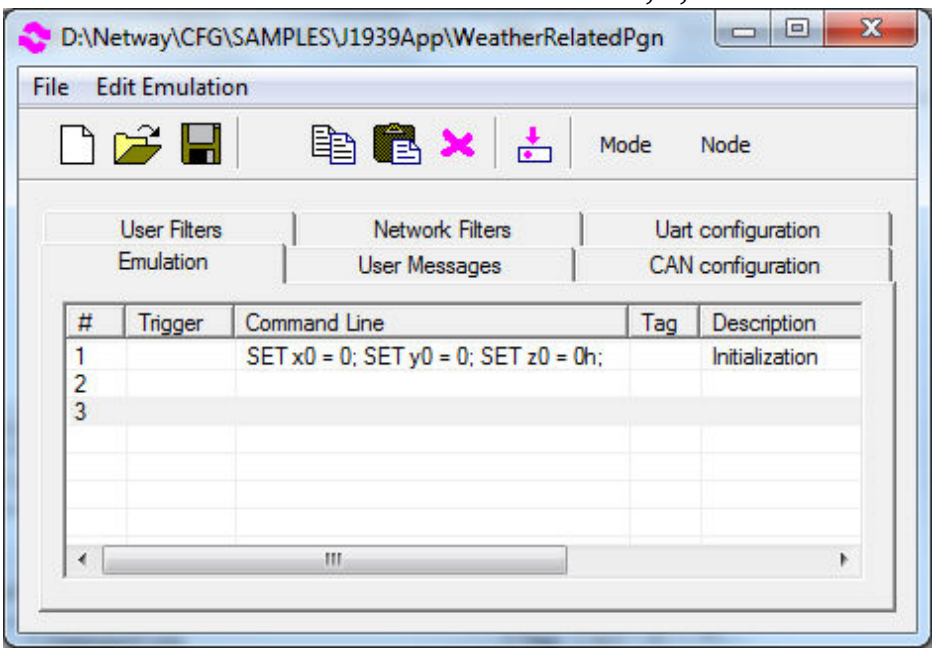
- **Note1:** Once data is set it will take effect when frame is scheduled for transmission in next cycle.
- **Note2:** Data set commands (SET linFrame) will take effect only if selected node is the owner of data.

Triggering command upon reception of LIN frames: Examples of command lines triggered by frames in command lines 7 and 11:
CL7: linFrame1 -> SET x10 = rxdata1; SET x11 = rxdata2; //Frame1,ch1 is received, store data
CL11: linFrame201 -> SET x20 = rxdata1; SET x21 = rxdata2; //Frame1,ch2 is received, store data

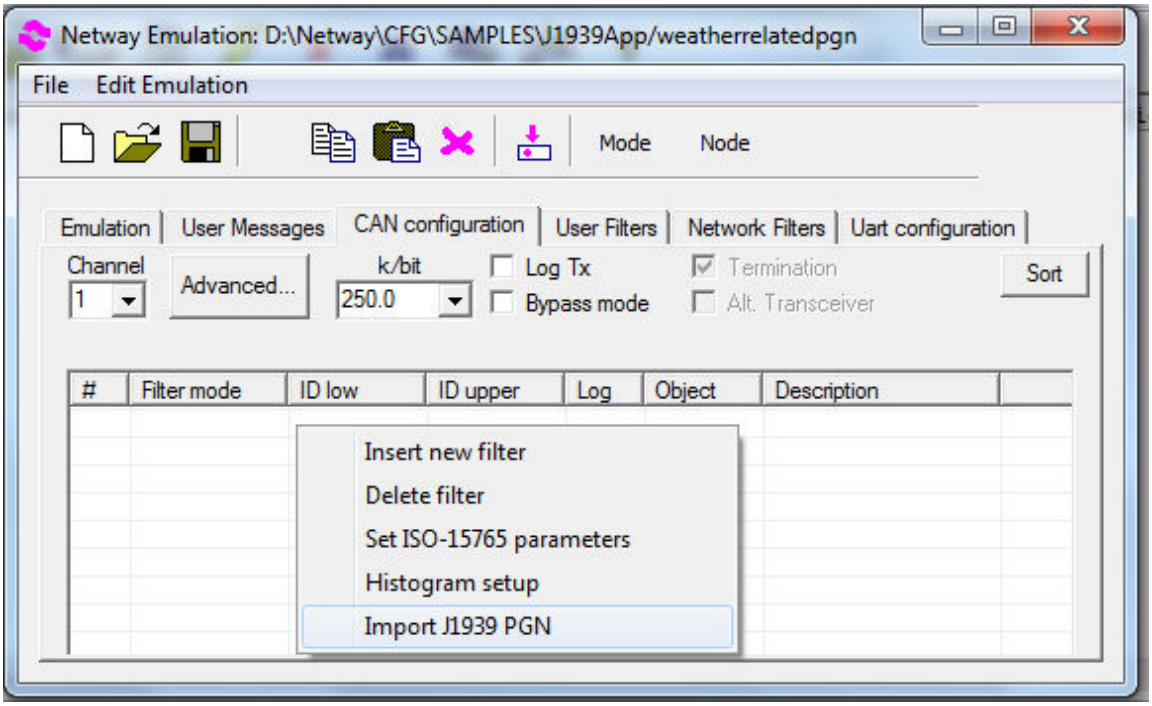
J1939

Example of how to configure Netway device for receiving specific parameters from J1939 bus is provided below:

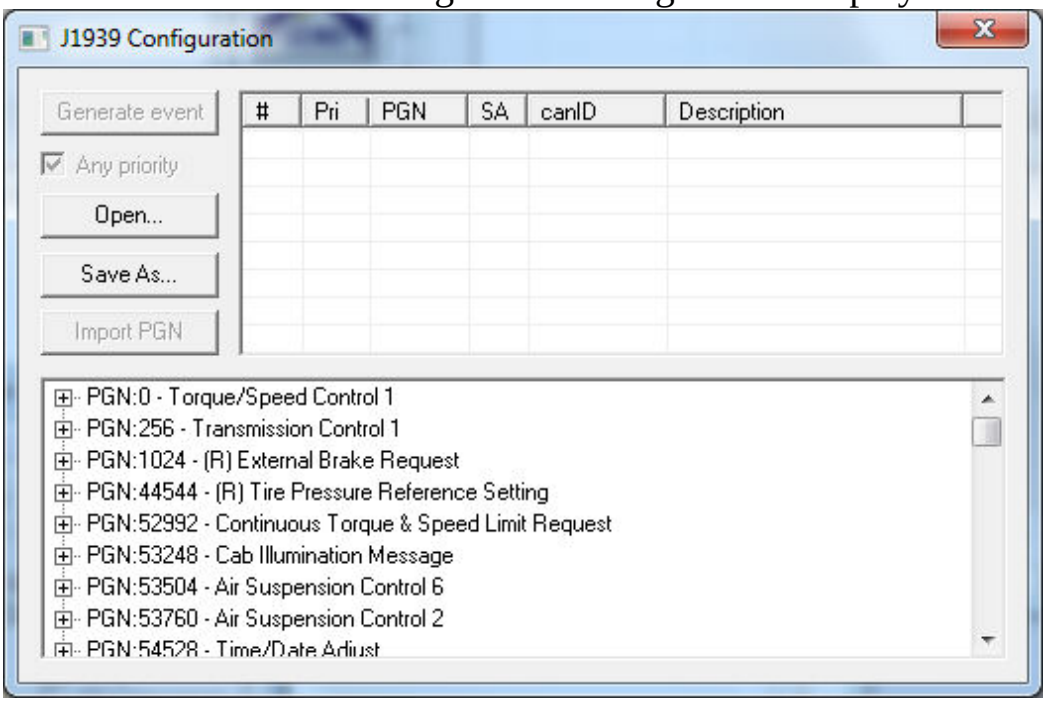
Step 1 - Initialization
Create a new emulation with default setting. Let's give it a name: "WeatherRelatedPgn.emu".
In the first command line we initialize X,Y,Z variables to 0 as show below:



Step 2 - Invoke J1939 configuration setup
Configure CAN channel 1 with 250.000 kbit/sec speed:

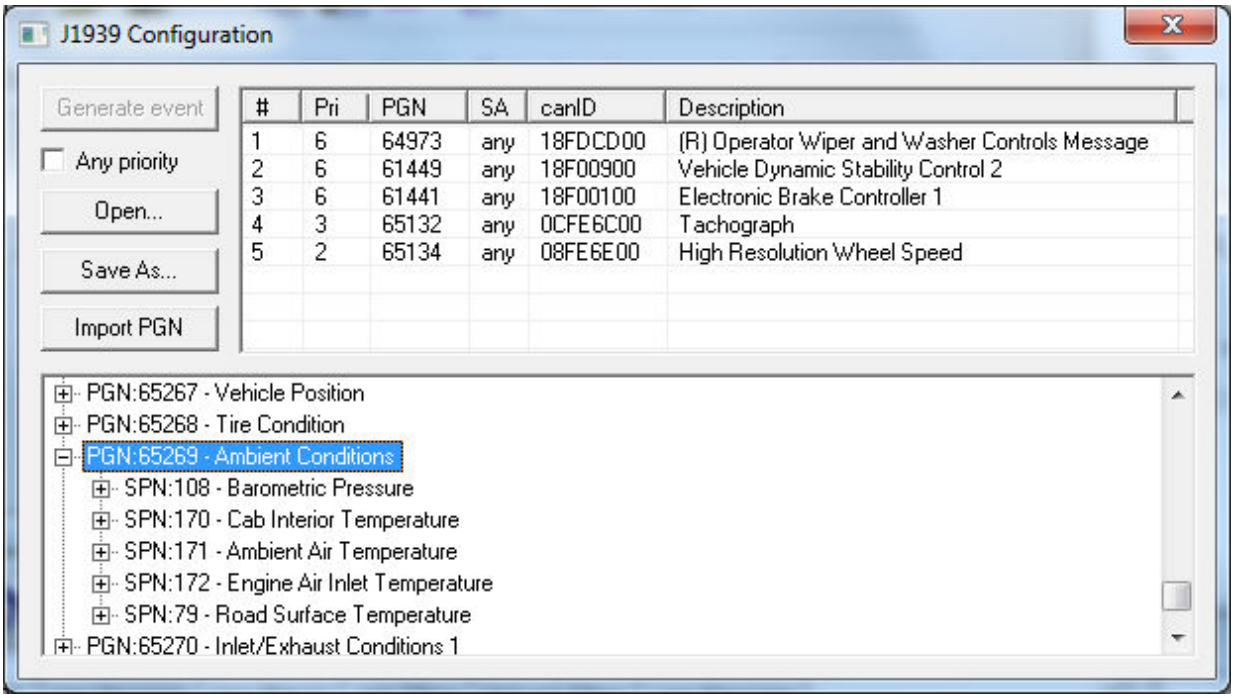


Right-click on CAN configuration page and select <Import J1939 PGN> command -the J1939 Configuration dialog will be displayed:



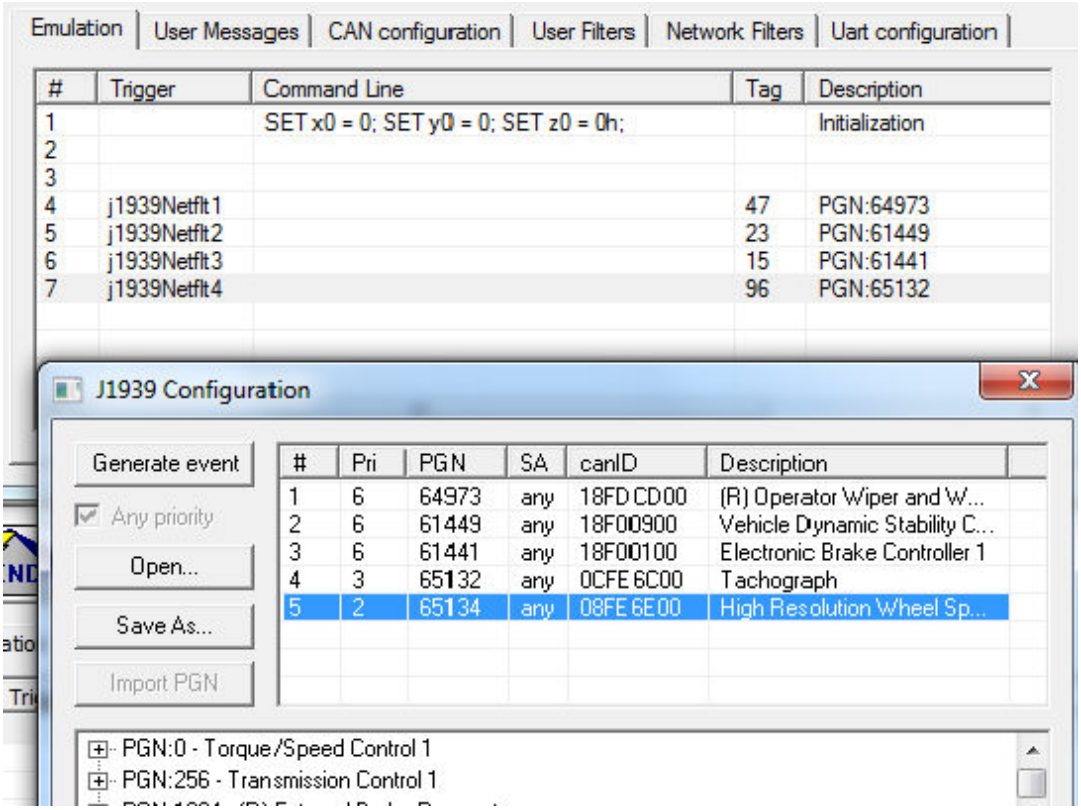
Step 3 - Importing desired PGN messages

Select relevant PGN from the tree list and import them into configuration list by clicking <Import PGN> button as show below:

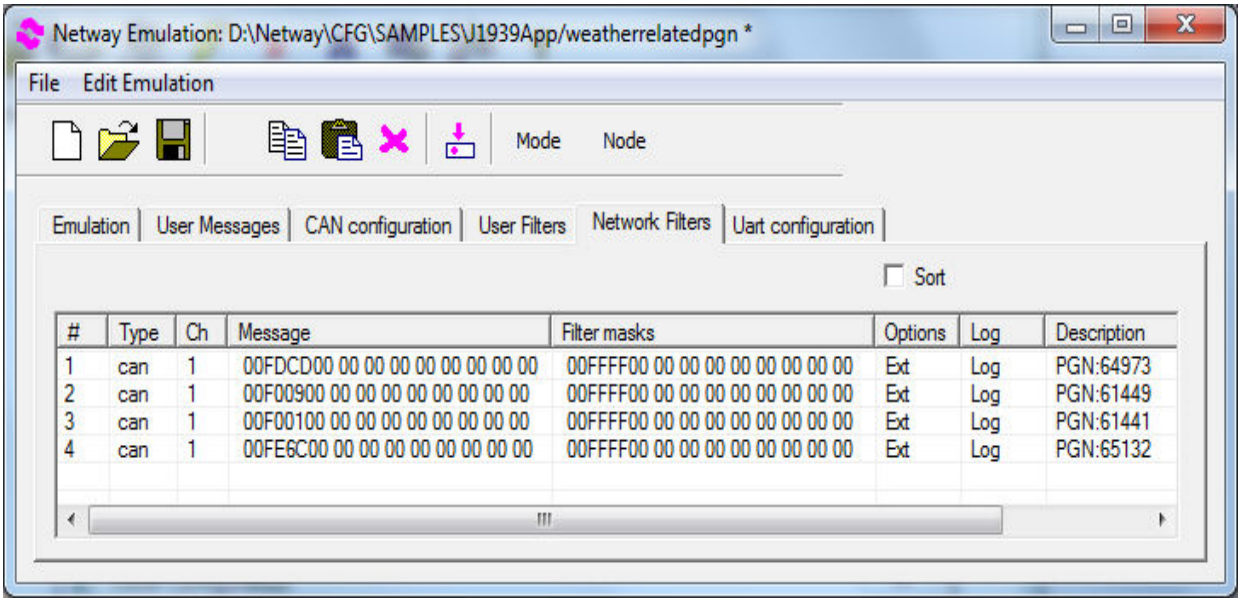

J1939 Configuration
Generate event
Any priority
Open...
Save As...
Import PGN
Pri PGN SA canID Description
1 6 64973 any 18FDCD00 (R) Operator Wiper and Washer Controls Message
2 6 61449 any 18F00900 Vehicle Dynamic Stability Control 2
3 6 61441 any 18F00100 Electronic Brake Controller 1
4 3 65132 any 0CFE6C00 Tachograph
5 2 65134 any 08FE6E00 High Resolution Wheel Speed
PGN:65267 - Vehicle Position
PGN:65268 - Tire Condition
PGN:65269 - Ambient Conditions
SPN:108 - Barometric Pressure
SPN:170 - Cab Interior Temperature
SPN:171 - Ambient Air Temperature
SPN:172 - Engine Air Inlet Temperature
SPN:79 - Road Surface Temperature
PGN:65270 - Inlet/Exhaust Conditions 1

Step 4 - Generating PGN events

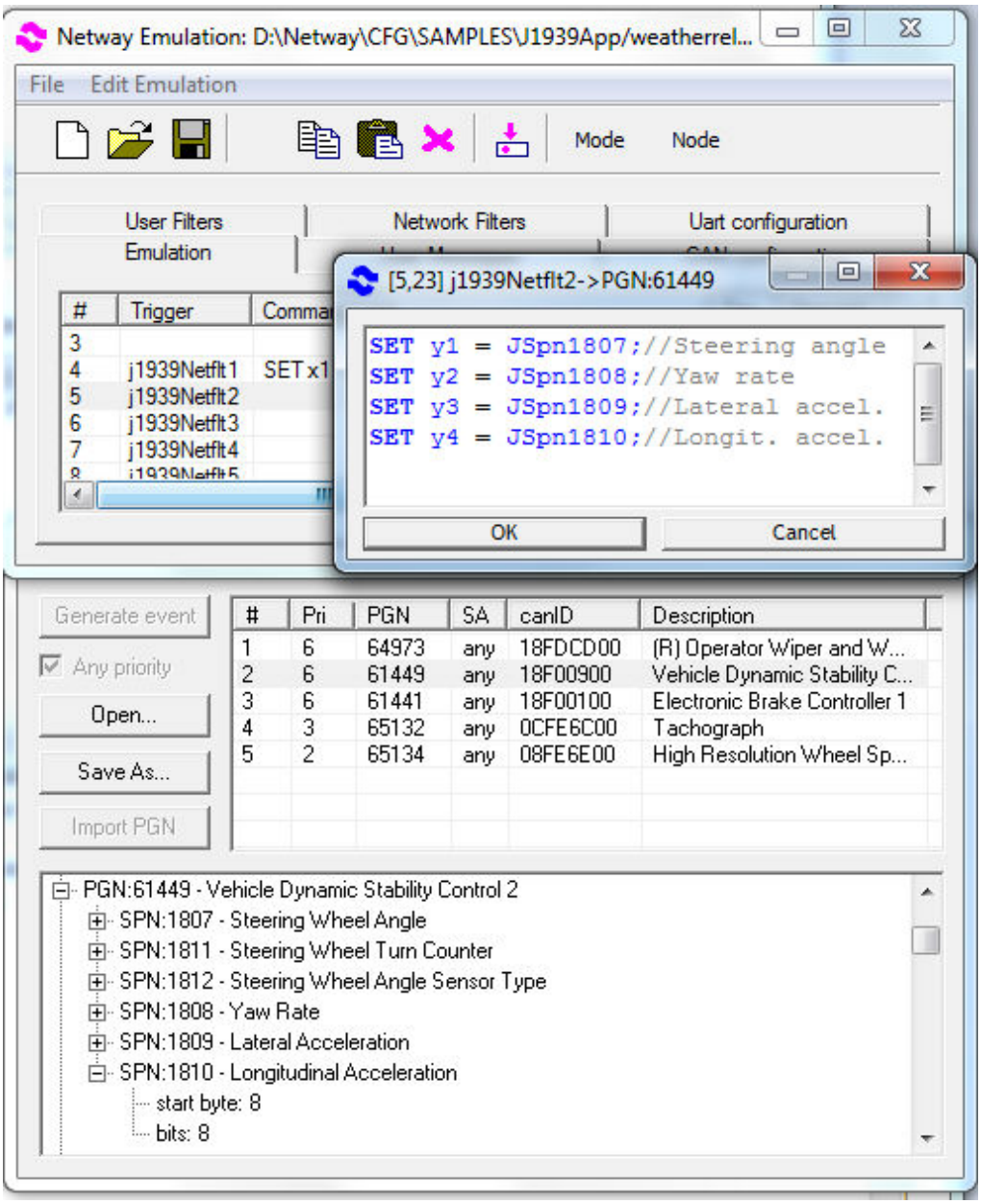
Generate PGN events by selecting imported PGN line and clicking on <Generate event> button:


Emulation
User Messages
CAN configuration
User Filters
Network Filters
Uart configuration
Trigger Command Line Tag Description
1 SET x0 = 0; SET y0 = 0; SET z0 = 0h; Initialization
2
3
4 j1939Netflt1 47 PGN:64973
5 j1939Netflt2 23 PGN:61449
6 j1939Netflt3 15 PGN:61441
7 j1939Netflt4 96 PGN:65132
J1939 Configuration
Generate event
Any priority
Open...
Save As...
Import PGN
Pri PGN SA canID Description
1 6 64973 any 18FD CD00 (R) Operator Wiper and W...
2 6 61449 any 18F00900 Vehicle Dynamic Stability C...
3 6 61441 any 18F00100 Electronic Brake Controller 1
4 3 65132 any 0CFE 6C00 Tachograph
5 2 65134 any 08FE 6E00 High Resolution Wheel Sp...
PGN:0 - Torque/Speed Control 1
PGN:256 - Transmission Control 1

Note, the emulation events are automatically generated with J1939NetfltN triggers. The command lines are empty at this point but they will be triggered upon receiving CAN message containing selected PGN. Network filters are also automatically generated, see them on Network Filters tab.


Netway Emulation: D:\Netway\CFG\SAMPLES\J1939App/weatherrelatedpgn *
File
Edit Emulation
Mode
Node
Emulation
User Messages
CAN configuration
User Filters
Network Filters
Uart configuration
Sort
Type Ch Message Filter masks Options Log Description
1 can 1 00FDCD00 00 00 00 00 00 00 00 00 00FFFF00 00 00 00 00 00 00 00 Ext Log PGN:64973
2 can 1 00F00900 00 00 00 00 00 00 00 00 00FFFF00 00 00 00 00 00 00 00 Ext Log PGN:61449
3 can 1 00F00100 00 00 00 00 00 00 00 00 00FFFF00 00 00 00 00 00 00 00 Ext Log PGN:61441
4 can 1 00FE6C00 00 00 00 00 00 00 00 00 00FFFF00 00 00 00 00 00 00 00 Ext Log PGN:65132

Step 5 - Customize command lines by assigning SPN to X,Y,Z
Customize command lines as desired by simply assigning SPN to a specific memory location (X,Y,Z variables):



Step 6 - More customizing of command lines

Let's say we want to adjust command line 4 (Wiper Operations) to accept data only when messages is sourced by non-zero address.(Least significant byte in CAN ID).

```
// get Source Address byte:
SET x1 = rxbyte3;
// if 0, do not except data:
IF x1 = 0;
        // end command line:
```

ENDCL;
ENDIF;
// Otherwise accept data
SET y21 = JSpn2865; // rear wiper SET y22 = JSpn2866; // front wiper
Similar customizing can be done for specific address or priority bits as desired.

Step 7 - More customizing of Network filters
One can narrow down acceptance of messages by editing acceptance mask for Network filter by setting one's or zero's in Filter mask column as shown below:
// Original Network filter 7: - accepts any ID mathing xxFEA9xx
can 1 00FEA900 00 00 00 00 00 00 00 00 00FFFF00 00 00 00 00 00 00 00 00
// Modified Network filter 7: - accepts ID only with SA = 12(hex)
can 1 00FEA912 00 00 00 00 00 00 00 00 00FFFFFF 00 00 00 00 00 00 00 00

Step 8 - Adding PGNs that are not listed by Netway application.
There is a special text file J1939.tbl located under Netway directory. Netway application is parsing this file to list all the PGNs and their SPNs.
If your J1939.tbl does not contain PGN that your application needs, you can modify the file in order to include desired PGN into the list as shown in example below:
Note, all PGN listed int the file in the ascending order. Suppose you need to add 64777 PGN into the table. The specification of a new PGN is shown below:

(R) PGN 64777 High Resolution Fuel Consumption (Liquid)

Engine fuel consumption accumulators. See PGN 65257 for alternate resolution.

Transmission Repetition Rate:	1 s	
Data Length:	8	
Extended Data Page:	0	
Data Page:	0	
PDU Format:	253	
PDU Specific:	09	PGN Supporting Information:
Default Priority:	6	
Parameter Group Number:	64777 (0x00FD09)	

Start Position	Length	Parameter Name	SPN
1-4	4 bytes	High Resolution Engine Trip Fuel	5053
5-8	4 bytes	High Resolution Engine Total Fuel Used	5054

1. Find the right place to insert the 64777 entry and insert the new record for 64777 PGN (shown in bold)

{61453,100,3,8,{ //(R) Land Leveling System Operational Information
{3156,0x11,4}, //Blade Control Mode Switch
{3157,0x15,4}, //Desired Grade Offset Switch
{3158,0x21,4}, //Blade Auto Mode Command
{0,0,0}}},
//PGN, Period(ms), default priority, length (bits)
{64777,1000,6,8,{ //(R) High Resolution Fuel Consumption (Liquid)
{5053,0x11,4}, //High Resolution Engine Trip Fuel - SPN 5053, starts at byte 1, bit 1, length 4 bits
{5054,0x15,4}, //High Resolution Engine Total Fuel Used - SPN 5054, starts at byte 1, bit 5, length 4 bits
{0,0,0}}},
{64945,500,6,8,{ //(R) After Treatment Intake Gas 2-b
{3275,1,16}, //Exhaust Gas Temperature 1 (bank 2)
{3276,3,16}, //Particulate Trap Intake Gas Temperature (bank 2)
{3277,0x51,5}, //Exhaust Gas Temperature 1 Preliminary FMI (bank 2)
{3278,0x61,5}, //Particulate Trap Intake Gas Temperature Preliminary FMI (bank 2)
{0,0,0}}},
{64777,1000,6,8,{ //(R) High Resolution Fuel Consumption (Liquid)
{5053,0x11,4}, //High Resolution Engine Trip Fuel
{5054,0x15,4}, //High Resolution Engine Total Fuel Used
{0,0,0}}},

J1939-DM1 service

With J1939 and J1708 protocols, the common task is to collect information of active faults reported by vehicle's various ECUs.
Netway software allows to configure the tool to automatically handle reported faults and/or report them back to host.
Abbreviations used in this chapter:
DMID - DM report identification DMID = 84h
LEN - DM report length (bytes)
DM - 16 memory structures allocated for monitoring 16 different ECUs DTC status **INX - index of reported DM (0-15 range)**
DM1 - diagnostics messages
SA, MID - source address of ECU sending DM. SA is used for J1939 messages, MID - for J1708 messages **PRID** - protocol ID of received DM (can1=3, can2=6 or j1708=14)
DTC - Data Trouble Code
nDTC - number of DTCs for reporting source address
LS, LS2 - lamp status (applicable for J1939 only)
MDCNT - modified count
RPID - report ID assigned by user
DLEN - Data length (bytes)

Processing diagnostic reports can be accomplished in two simple steps:
Step 1: Initialize DM1 processing (usually it is done in command line 1):
SET jDM0 = 65226;//Initialize DM1reports
Step 2: Trigger reports on any change by specifying an event with this trigger:
j1939DM1->SET reportHost700 = 5Ah; //DM report with custom ID 5A (hex)
or/and
j1708DM1->SET reportHost700 = 6Ah; //DM report with custom ID 6A (hex)

Device is capable to monitor faults on different protocols:CAN1/2(J1939) and UART2(J1708). The results are sorted by source address (SA) and protocol ID. Totally 16 ECUs can be monitored for any changes in faults

information. Once the change is detected, j1708DM1 or j1939DM1 triggers the corresponding command line.
Netway device is capable to monitor and detect DM1 reports from up to 16 ECUs identified by their source addresses (SA) and protocol. For each PRID/SA, it allocates DM memory structure to reflect any changes:
Examples:
1. **j1939DM1**->SET reportHost700 = 5Ah;
In this example multiple (8 DTCs) faults has been reported by SA=EAh on can1 network:
1 can1 1CECFFEA 20 22 00 05 FF CA FE 00
2 can1 1CEBFFEA 01 D7 FF A7 0F 1F 63 80
3 can1 1CEBFFEA 02 06 1F 05 6E 00 04 63
4 can1 1CEBFFEA 03 64 00 04 63 60 00 11
5 can1 1CEBFFEA 04 63 C2 02 1F 01 0A 0F
6 can1 1CEBFFEA 05 1F 01 B1 0F 1F 01 FF
DM1 will be detected and the command line above will be triggered to report the change in DTC status. The following report will be generated(hex):
DMID LEN RPID PRID/INX SA LS nDTC MDCNT DTC1 DTC2 DTC3 DTC4 DTC5 DTC6 DTC7 DTC8
84 28 6A 30 EA D7 08 01 A70F1F63 80061F05 6E000463 64000463 60001163 C2021F01 0A0F1F01 B10F1F01

2. **j1708DM1**->SET reportHost700 = 6Ah; //DM report with custom ID 6A (hex)
In this example 3 DTCs has been reported by MID=80h on j1708 network:
j1708 80 C2 0C 6F E3 01 F8 F2 01 69 E0 01 6E E0 EE
DM1 will be detected and the command line above will be triggered to report the change in DTC status. The following report will be generated (hex):

DMID LEN RPID PRID/INX MID DLEN nDTC MDCNT DTC1 DTC2 DTC3 DTC4
84 14 6A E1 80 0C 04 01 6FE301 F8F201 69E001 6EE0EE

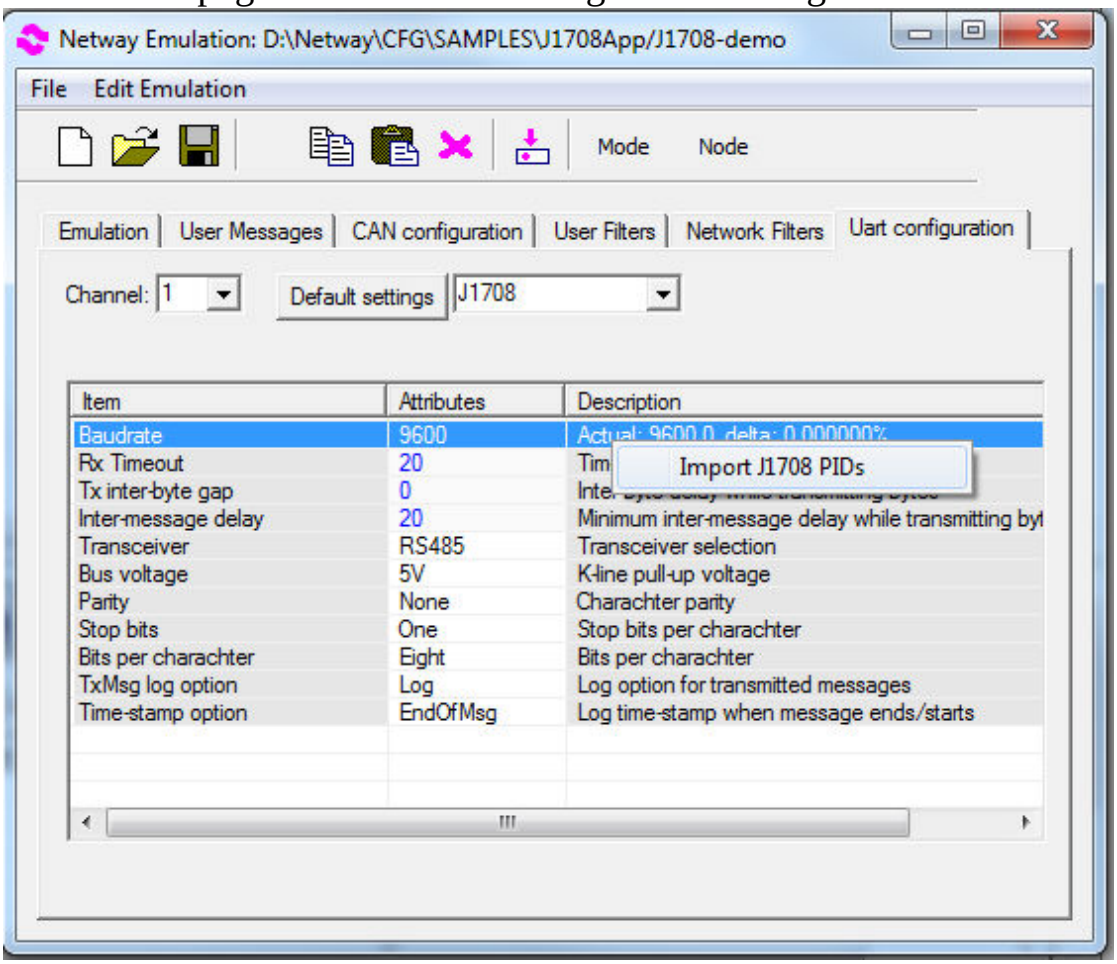
Note: In PRID/INX byte - protocol ID is in most significant 4 bits and DM index in 4-least significant bits. Index 0-15 range covers all 16 SA detected by firmware

Step 3: Sometimes truck generates frequent DM1 changes that leads to frequent DM1 reports. User can limit the frequency of reports by filtering them by time or ignoring reports due to lamp status change. In order to accomplish the filtering consider examples below:
After assigning all SPNs/PGNs user has to use this command to enable active SPN table: **ENABLE JPgn0;**
To enable DM1 filtering for J1939 bus, one can use the following:
ENABLE JPgn51;
Resulting in minimum timing for reporting 5 seconds, ignore lamp status change (LS-Ignore is ON)
ENABLE JPgn20**;**
Resulting in minimum timing for reporting 2 seconds, include lamp status change (LS-Ignore is OFF)
ENABLE JPgn0; command disables filtering
Important note: SET jDM0 = 65226; must be used prior to setting DM1 filters ENABLE JpgnXY;
Step 4: When needed, user can select DM index to be reported in few ways:
1. SET jDM1 = n;//n - desired index n = 0 -:- nDTC-1
For example SET jDM1 = 1; will assign reported index to 1, then SET reportHost700 = 5Ah; will report second DM
2. SET reportHost701 = 5Bh; this command will move the current index to 1 and report it
SET reportHost702 = 5Ch; this command will move the current index to 2 and report it
3. Set of following command will search for DM with given SA and protocol: SET y1 = 03EAh;//find can1 EAh SA
IF jDM2 = y1;//if found, report it
SET reportHost700 = 3Ah;
ENDIF;
NOTE: Current DM index changes automatically when any DM change event is detected. SET reportHost700 = N; always reports current DM index which can be set manually as discussed below.

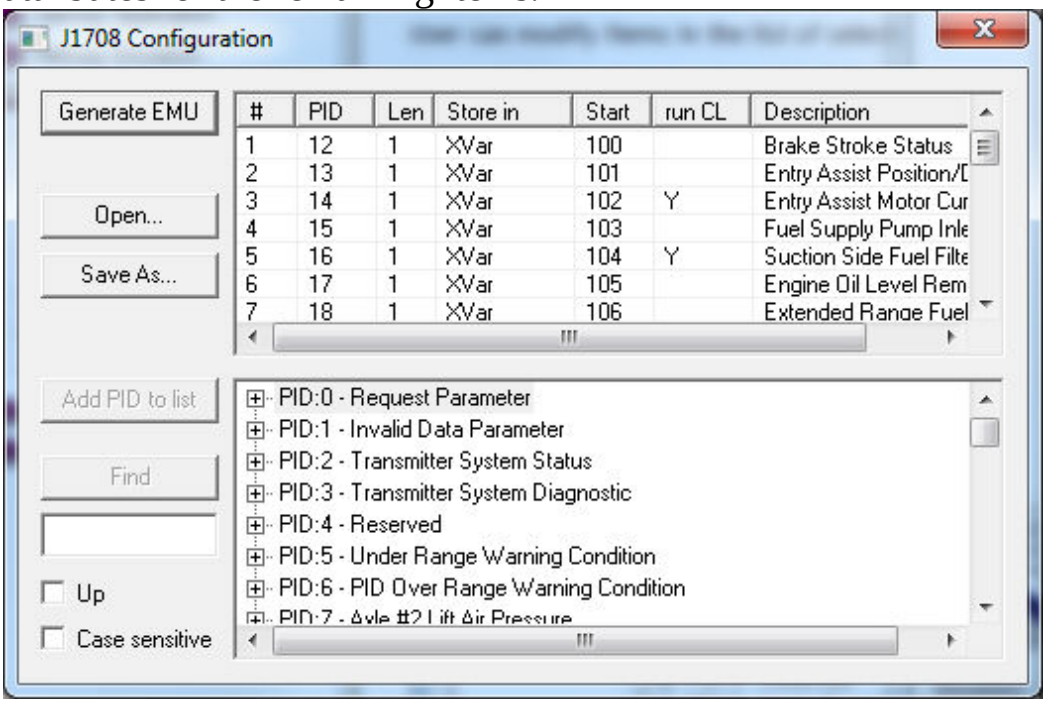
Additional commands are available for user to analyze content of current DM:
SET xN = jDM1;//copies DM to x-variables xN to xM, M=N+4*nDTC for CAN and M=N+DLEN for J1708
SET xN = jDM2; //source address (SA/MID)
SET xN = jDM3; //MDCNT number of times DM been modified
SET xN = jDM4; //number of DTCs for this SA/SID
SET xN = jDM5; //lamps status for CAN protocols, for j1708 it is DLEN
SET xN = jDM6; //number of detected ECUs (SA/MID and PRID)
SET xN = jDM7; //DM number (current index + 1)
SET xN = jDM8;//lamp status 2 for CAN protocols, reserved for J1708
SET xN = jDM9;//PRID protocol ID for current DM (can1=3, can2=6, j1708=14)

J1708

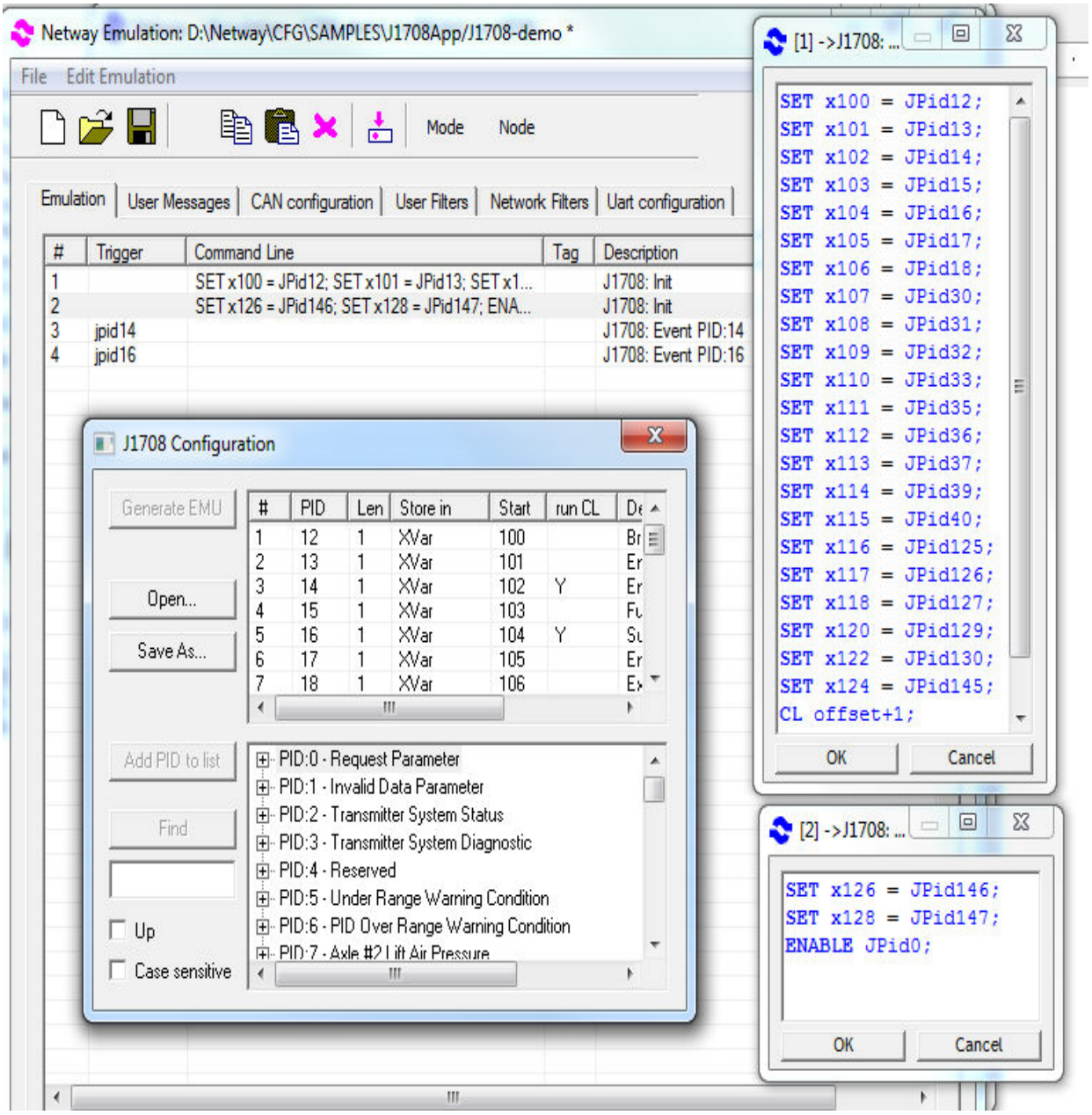
J1708 protocol type can be selected as one of the Netway featured protocols in the *UART* dialog. Any of the *UART* channels can be setup to work according to the *J1708* protocol assuming the selected channel has RS485 transceiver.
Users can read more details on the *J1708* standard in *SAE* related documents. J1587 SAE standard defines *PID*'s and *MID*'s for all vehicle signals.
Netway tool takes advantages on the fact that all vehicle signals are well defined in *J1708* protocol via *PID*'s (Parameter ID) and *MID*'s (Message ID).
Once J1708 protocol is selected in Uart configuration property page, right-click on the page to start J1708 configuration dialog as shown below:



The J1708 configuration dialog displays all the J1708 PIDs in lower portion of dialog.
Upper portion of the dialog contains list of user selected PIDs.
To Add PID to the list, user need to highlight tree item and click on <Add PID to list> button.
To delete item from the list, highlight the list item and press <Delete> key.
User can modify items in the list of selected PIDs by double-clicking on <Store in>, <Start> and <run CL> columns of the corresponding item.
Note, when <run CL> is selected to "Y", selected PID will generate a command line to perform PID specific actions if necessary.
When <Start> attribute for the first item is changed, it'll change all Start attributes for the remaining items.


J1708 Configuration
Generate EMU
Open...
Save As...
Add PID to list
Find
Up
Case sensitive
PID Len Store in Start run CL Description
1 12 1 XVar 100 Brake Stroke Status
2 13 1 XVar 101 Entry Assist Position/
3 14 1 XVar 102 Y Entry Assist Motor Cur
4 15 1 XVar 103 Fuel Supply Pump Inle
5 16 1 XVar 104 Y Suction Side Fuel Filte
6 17 1 XVar 105 Engine Oil Level Rem
7 18 1 XVar 106 Extended Range Fuel
PID:0 - Request Parameter
PID:1 - Invalid Data Parameter
PID:2 - Transmitter System Status
PID:3 - Transmitter System Diagnostic
PID:4 - Reserved
PID:5 - Under Range Warning Condition
PID:6 - PID Over Range Warning Condition

Once selection is complete click on <Generate EMU> button, to generate all command lines for processing selected PIDs.
Result if such action is shown below:


Netway Emulation: D:\Netway\CFG\SAMPLES\J1708App/J1708-demo *
File Edit Emulation
Mode Node
Emulation User Messages CAN configuration User Filters Network Filters Uart configuration
Trigger Command Line Tag Description
1 SET x100 = JPid12; SET x101 = JPid13; SET x1... J1708: Init
2 SET x126 = JPid146; SET x128 = JPid147; ENA... J1708: Init
3 jpid14 J1708: Event PID:14
4 jpid16 J1708: Event PID:16
J1708 Configuration
Generate EMU
Open...
Save As...
Add PID to list
Find
Up
Case sensitive
PID Len Store in Start run CL
1 12 1 XVar 100
2 13 1 XVar 101
3 14 1 XVar 102 Y
4 15 1 XVar 103
5 16 1 XVar 104 Y
6 17 1 XVar 105
7 18 1 XVar 106
PID:0 - Request Parameter
PID:1 - Invalid Data Parameter
PID:2 - Transmitter System Status
PID:3 - Transmitter System Diagnostic
PID:4 - Reserved
PID:5 - Under Range Warning Condition
PID:6 - PID Over Range Warning Condition
[1] ->J1708: ...
SET x100 = JPid12;
SET x101 = JPid13;
SET x102 = JPid14;
SET x103 = JPid15;
SET x104 = JPid16;
SET x105 = JPid17;
SET x106 = JPid18;
SET x107 = JPid30;
SET x108 = JPid31;
SET x109 = JPid32;
SET x110 = JPid33;
SET x111 = JPid35;
SET x112 = JPid36;
SET x113 = JPid37;
SET x114 = JPid39;
SET x115 = JPid40;
SET x116 = JPid125;
SET x117 = JPid126;
SET x118 = JPid127;
SET x120 = JPid129;
SET x122 = JPid130;
SET x124 = JPid145;
CL offset+1;
OK Cancel
[2] ->J1708: ...
SET x126 = JPid146;
SET x128 = JPid147;
ENABLE JPid0;
OK Cancel

Netway application automatically generates assignments for all PIDs, for example:

SET x126 = JPid146 // x126,x127 variables are allocated for PID 146. The PID 146 value will be stored there automatically by firmware when the PID is received.

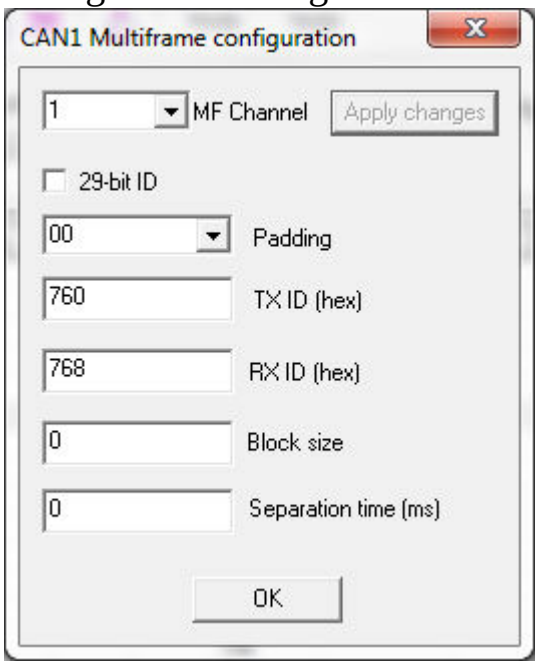
ENABLE JPid0; // this command is necessary to initialize all PIDs reception and processing and it must appear after all PIDs assignments are performed.

jpid14,15 triggers are generated because <run CL> attributes were selected with "Y" option for PID 14 and 16

CAN Multi-frame

Note: This section assumes that a user is familiar with Multi-Frame CAN (ISO-15765) protocol specifications.
The Multi-Frame CAN (ISO-15765) protocol can be utilized on any CAN channels of a Netway device, however availability of this protocol is determined by the Premium Options defined for that Netway device. Therefore a Netway device compatible with the Multi-Frame CAN (ISO-15765) protocol will have the ISO-15765 (CanMultiFrame) Checkbox checked in the Premium Options Window and the About Netway Window will display the following line: **<Premium Option: ISO-15765 (CanMultiFrame)>**.

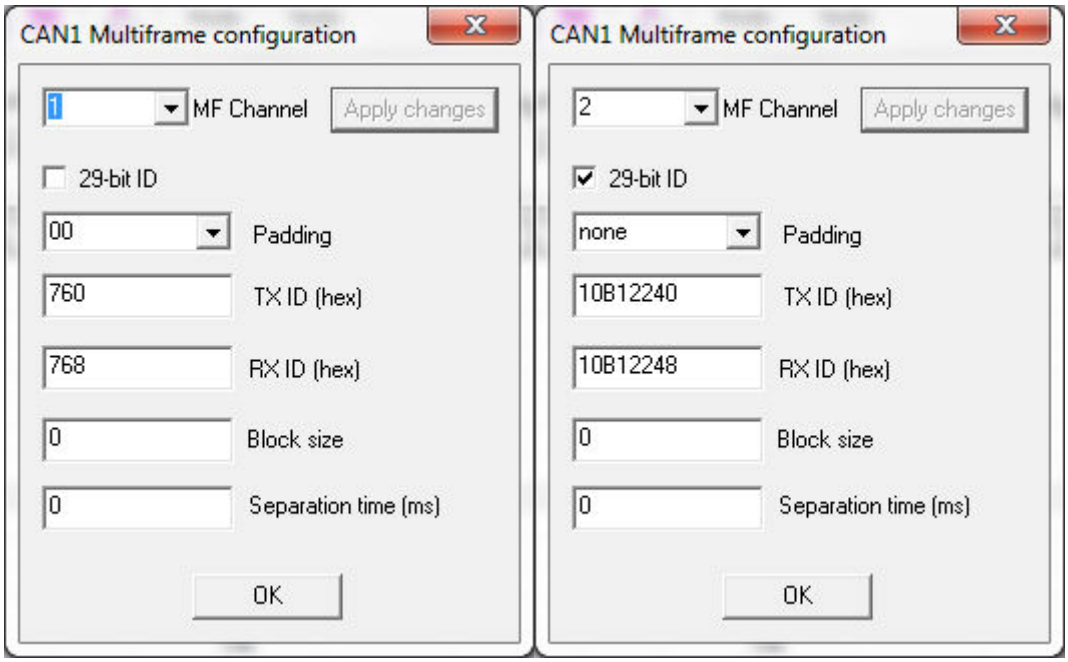
1. ISO-15765 configuration
CAN configuration tab of Emulation dialog allows user to configure ISO-15765 for each channel.
Right-click and selecting <Set ISO-15765 parameters> command to invoke configuration dialog:



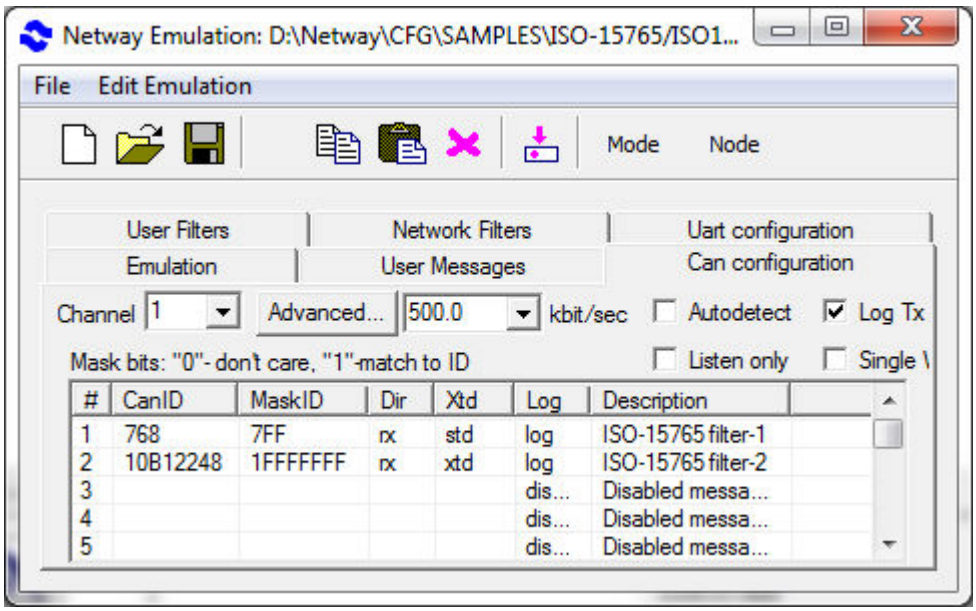
User can configure up to four different MF channels per each CAN channel. When changes are made for a given MF channel, click on <Apply changes> button before switching to the next MF channel.
In order to disable MF channel enter 0 to TX ID field and RX ID field.

Once all needed MF channels are configured click <OK> button.
ISO-15765 filters will be automatically inserted in CAN configuration page.

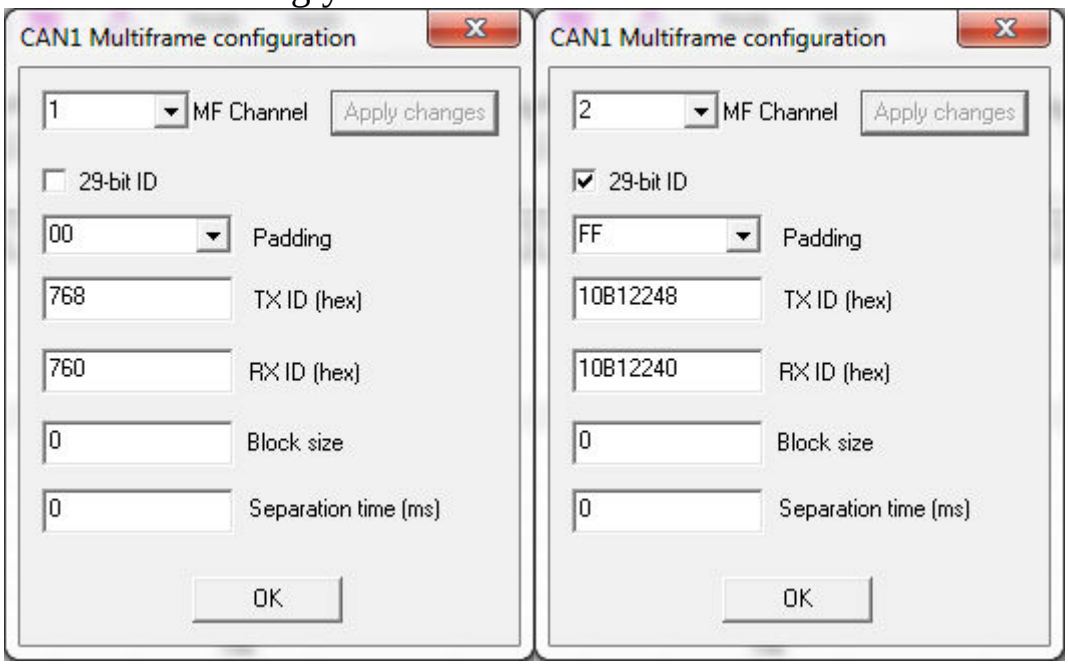
2. Using Multi-frame configuration
Two examples of how to use multi-frame configuration are available in SAMPLES/ISO-15765 directory.
Load first Netway device with ISO15765_Node1.emu and second device with ISO15765_Node2.emu
CAN channel 1 ISO-15765 for first device is configured for two MF channels as follows:



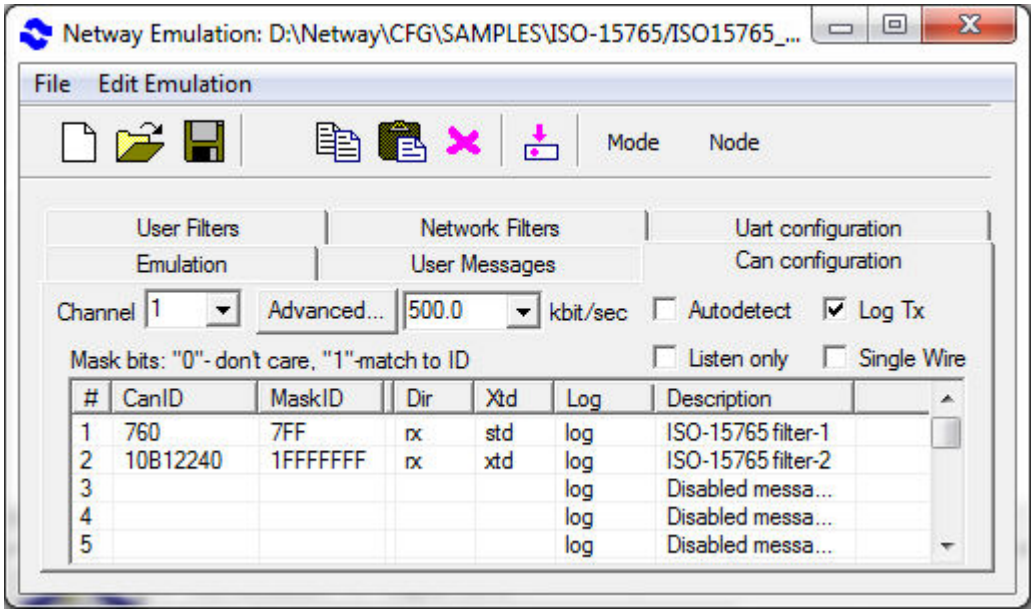
After click on <OK> button multi-frame filters are automatically inserted in CAN configuration page as shown below:



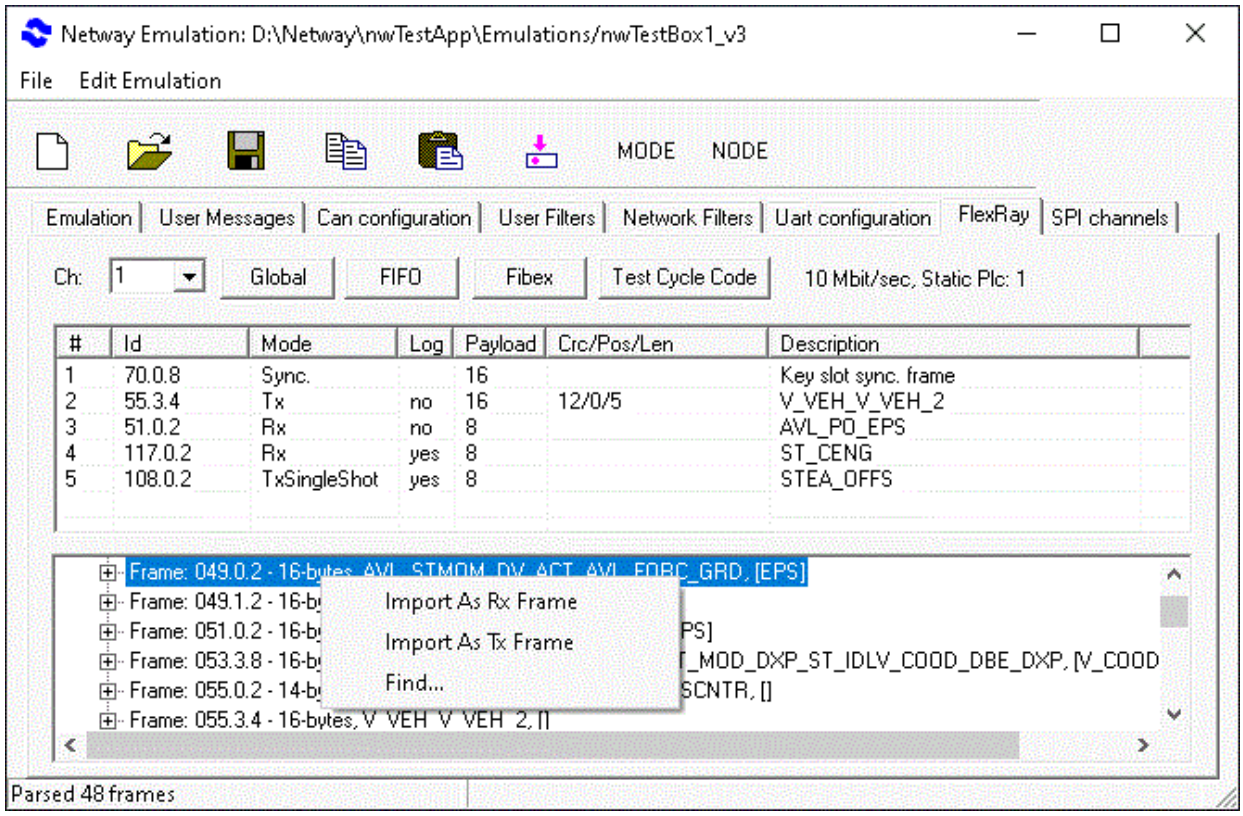
CAN channel 1 ISO-15765 for second device is also configured for two MF channels accordingly to match the first device:



After click on <OK> button multi-frame filters are automatically inserted in CAN configuration page as shown below:


Netway Emulation: D:\Netway\CFG\SAMPLES\ISO-15765/ISO15765_...
File
Edit Emulation
Mode
Node
User Filters
Network Filters
Uart configuration
Emulation
User Messages
Can configuration
Channel 1
Advanced...
500.0
kbit/sec
Autodetect
Log Tx
Mask bits: "0"- don't care, "1"-match to ID
Listen only
Single Wire
CanID MaskID Dir Xtd Log Description
1 760 7FF rx std log ISO-15765 filter-1
2 10B12240 1FFFFFFF rx xtd log ISO-15765 filter-2
3 log Disabled messa...
4 log Disabled messa...
5 log Disabled messa...

Flexray



Controls:

- **Ch** - channel number 1-:-3
- **<Global>** - global Flexray parameters per manufacturer spec
- **<FIFO>** -
- **<Fibex>** - file to use for importing frames

Flexray configuration list:

- **#** - object/frame number to be used as a reference in emulation
- **Id** - Flexray frame Id
- **Mode** - Tx/Rx/Sync.
- **Log** - log/discard
- **Payload** - imported data bytes length
- **CRC/Pos/Len** - some imported frames has PDUs with CRC data protection

- **Description** - Imported with all frame parameters from Flexray

Right-click on configuration list displays available command:

- Delete Item
- Open configuration
- Save configuration

Fibex Import tree:

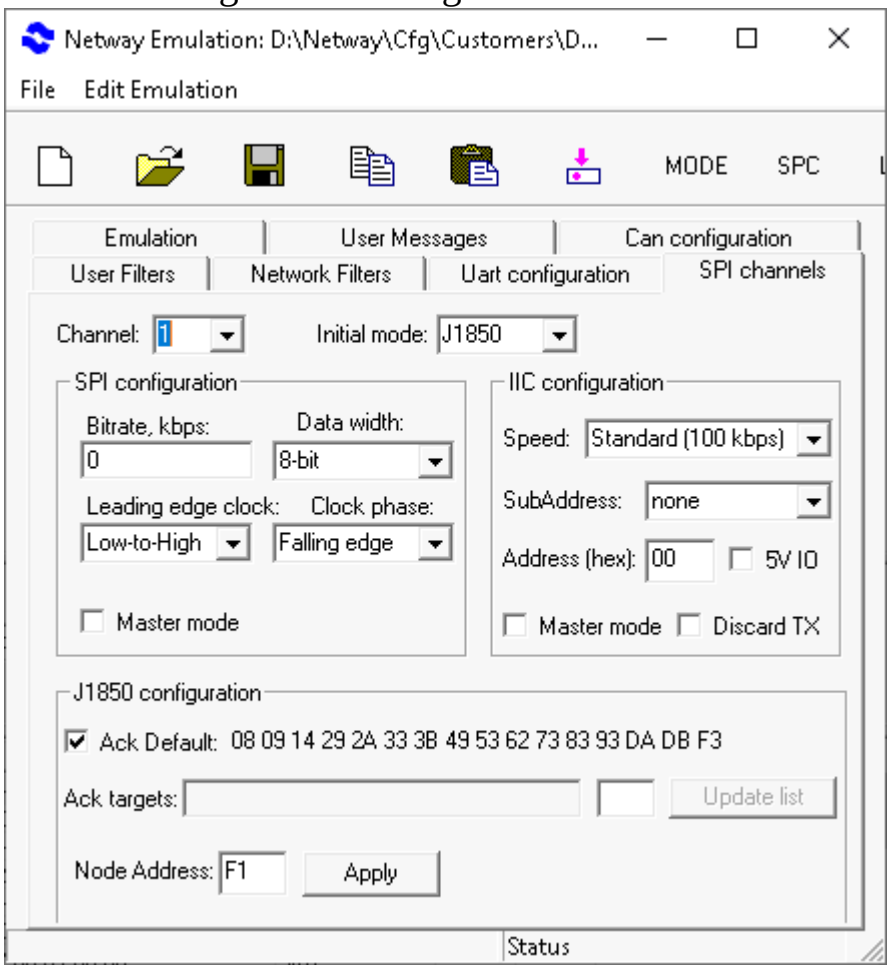
Right-click on Fibex tree item to display menu commands:

- Import As Rx Frame
- Import As Tx Frame

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

SPI/IIC/J1850 Channels

SPI DB configuration dialog is shown below:



- **Channel** - channel numbers 1 or 2
- **Initial mode** - none / SPI / I2C / J1850

Note: mode can be changed "on fly" by emulation commands.

SPI configuration

- **Bitrate, kbps** - up to 1000 kbps
- **Data width** - 8 / 16 bits
- **Leading edge clock** - Low to high / High to low
- **Clock phase** - Falling edge / Rising edge
- **Master mode** - Master / Slave select

IIC configuration

- **Speed** - Fast (400k) / Standard (100k)
- **Sub-address** - none / 1 / 2 / 3 / 4
- **Address** - for slave mode
- **5V IO** - 5V/3V
- **Master mode** - Master / Slave select
- **Discard TX** - discard / log

J1850 configuration

Two SCP (PWM) or Class2(VPW) channels.
Each channel is able to Tx and/or Rx frames.
PWM/VPW modes are automatic

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

CAN FD daughter board

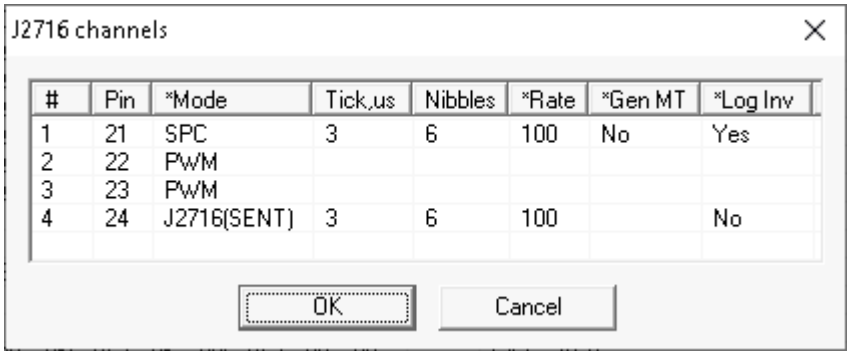
Type topic text here.

J2716 SENT/SPC

Four I/O J2716 channels available:

- Four SENT Inputs channels; Pins 21,22,23,24
- Four SENT Outputs channels: Pins 16,17,18,19

Emulation/MODE/J2716 Configuration command brings forward dialog for J2716 channels:


J2716 channels
Pin *Mode Tick,us Nibbles *Rate *Gen MT *Log Inv
1 21 SPC 3 6 100 No Yes
2 22 PWM
3 23 PWM
4 24 J2716(SENT) 3 6 100 No
OK
Cancel

- *Mode I/O / J2716 / SPC
- Ticks,us: Default value 3 microseconds per tick
- Nibbles: Default value 6 data nibble + 1 status nibble + 1 CRC nibble
- *Rate: Update rate in milliseconds
- Gen MT: Generate Master trigger, specify "No" if monitoring existing SPC system
- *LogInv: log messages with invalid CRC

SENT channel pins require a pull-up resistor (2k-10k) to 5V. Pin 43 can be used as a source of 5V.
<*Mode>, <*Rate> and <logInv> columns can be edited by user.

If *Mode=J2617(SENT), the input is processed as SENT input, otherwise it is processed as PWM input
Corresponding to channel output pin is processed as PWM output.

Monitoring for Programmable Linear Hall Sensor TLE4998S3C(SENT) connected to pin 24 with 4.7k pull-up:

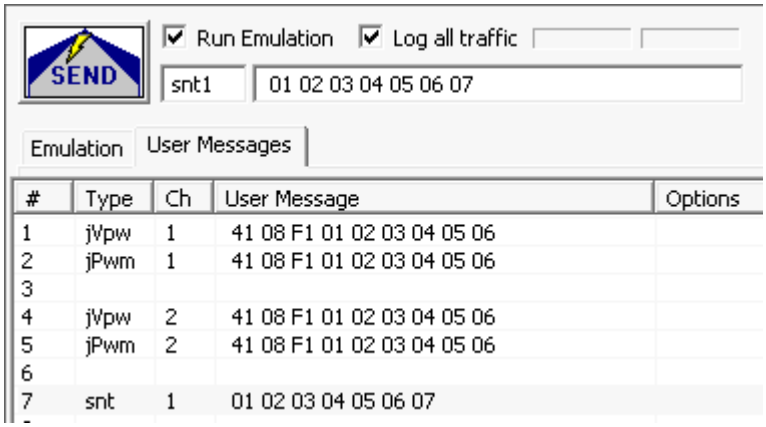
No Master trigger is generated in this case, not needed for SENT protocol.

Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame
1	0:45:11.032	snt4		8	01 08 00 04 08 05 04 01
2	0:45:11.132	snt4		8	01 08 00 03 09 05 04 07
3	0:45:11.232	snt4		8	01 08 00 04 02 05 04 0D
4	0:45:11.332	snt4		8	01 08 00 04 02 05 04 0D
5	0:45:11.532	snt4		8	01 08 00 04 08 05 04 01
6	0:45:11.632	snt4		8	01 08 00 04 08 05 04 01
7	0:45:11.732	snt4		8	01 08 00 04 08 05 04 01
8	0:45:11.832	snt4		8	01 08 00 03 0F 05 04 08
9	0:45:11.932	snt4		8	01 08 00 04 02 05 04 0D
10	0:45:12.032	snt4		8	01 08 00 04 0B 05 04 08
11	0:45:12.132	snt4		8	01 08 00 04 02 05 04 0D
12	0:45:12.232	snt4		8	01 08 00 04 02 05 04 0D
13	0:45:12.332	snt4		8	01 08 00 04 08 05 04 01
14	0:45:12.432	snt4		8	01 08 00 04 08 05 04 01
15	0:45:12.532	snt4		8	01 08 00 04 08 05 04 01

Monitoring for Programmable Linear Hall Sensor TLE4998C8(SPC) connected to pin 24 with 4.7k pull-up:
Master trigger is generated (6 uS low) by this tool every 100ms to retrieve data from sensor

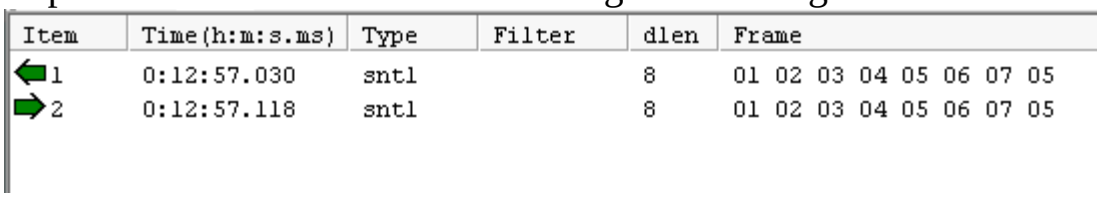
Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame
1	0:55:28.932	sntl		8	03 08 00 01 02 05 02 0F
2	0:55:29.032	sntl		8	03 08 00 01 02 05 02 0F
3	0:55:29.132	sntl		8	03 08 00 00 00 05 02 05
4	0:55:29.232	sntl		8	03 08 00 00 0C 05 02 06
5	0:55:29.332	sntl		8	03 08 00 00 03 05 02 0C
6	0:55:29.432	sntl		8	03 08 00 00 09 05 02 00
7	0:55:29.532	sntl		8	03 08 00 00 03 05 02 0C
8	0:55:29.632	sntl		8	03 08 00 00 09 05 02 00
9	0:55:29.732	sntl		8	03 08 00 01 02 05 02 0F
10	0:55:29.832	sntl		8	03 08 00 00 00 05 02 05
11	0:55:29.932	sntl		8	03 08 00 01 02 05 02 0F
12	0:55:30.032	sntl		8	03 08 00 01 05 05 02 07
13	0:55:30.132	sntl		8	03 08 00 00 09 05 02 00
14	0:55:30.232	sntl		8	03 08 00 00 09 05 03 0D
15	0:55:30.332	sntl		8	03 08 00 00 0C 05 03 0B

Transmitting SENT message via output channel:



If pin SENT Outputs channel 1 (pin 16) is connected to SENT Inputs channel 1 (pin 21)

Capture below shows result of sending user message7:

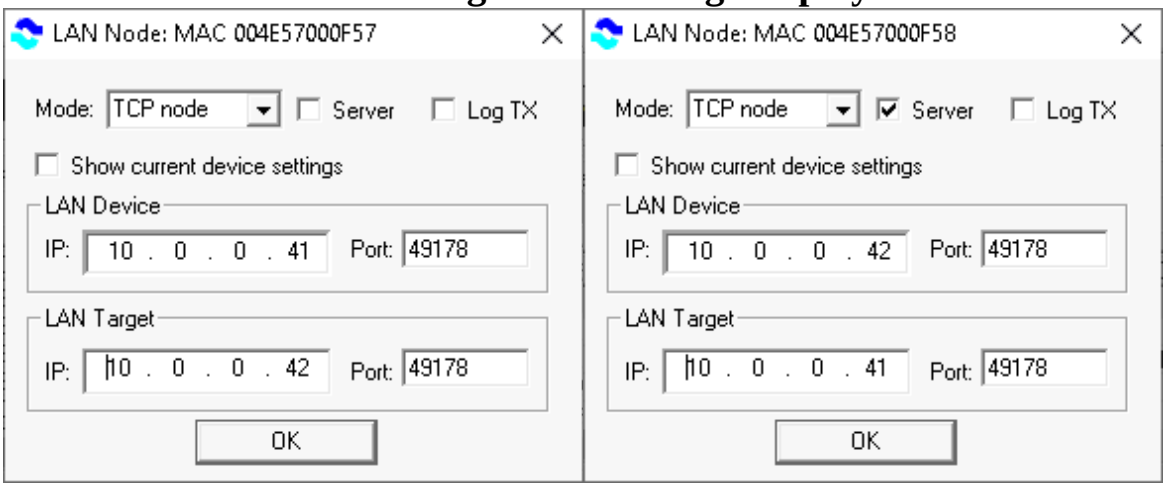


LAN / Ethernet Communications

When user clicks on Emulation->LAN button - the LAN configuration is displayed.
Netway device operates on LAN in two modes: TCP or UDP.
TCP mode requires external DHCP server, two nodes must communicate by server/client scheme.
UDP mode does not need external DHCP, two nodes can be connected by standard LAN wire to communicate.

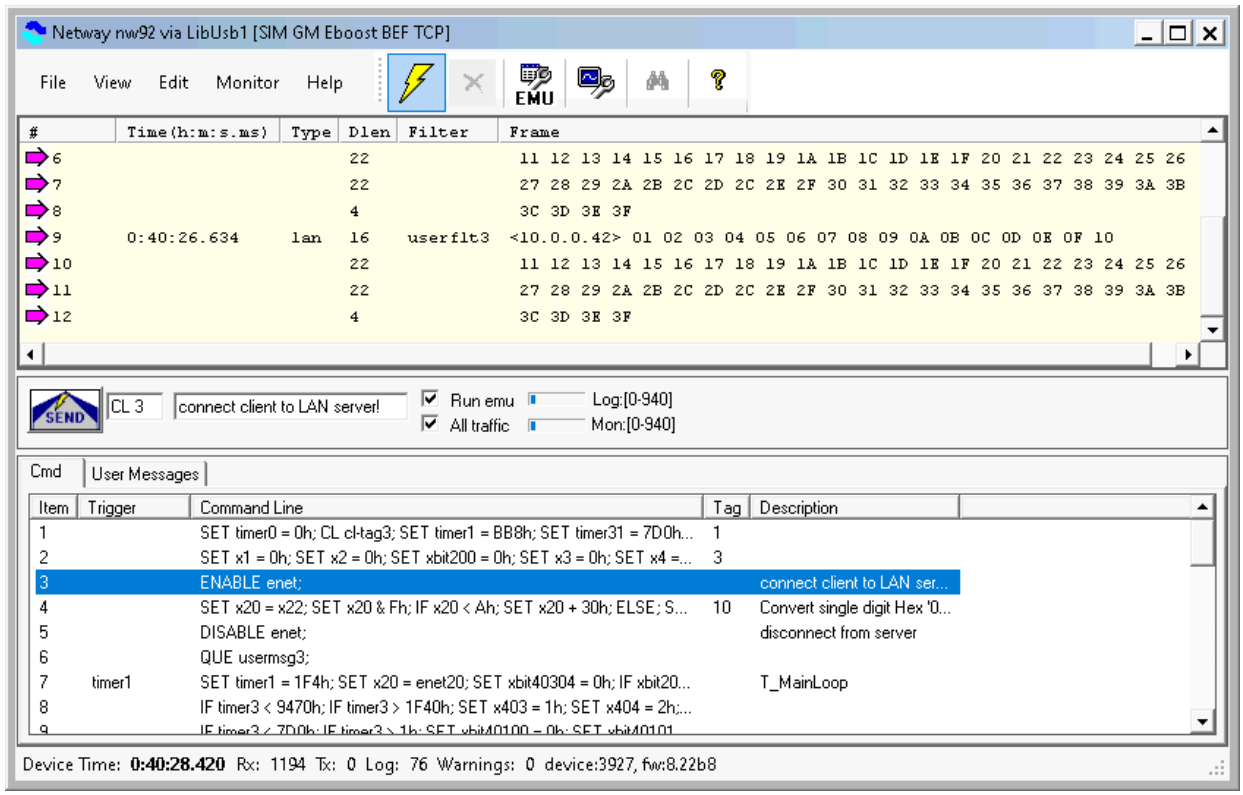
TCP mode:

Once powered-up device acquires it's IP from DHCP with assigned IP=10.0.0.42 for server node and 10.0.0.41 for client node.
Devices operate as server (listening) or client node. Client initializes connection to server in order to communicate.
For server/client node configuration dialogs displayed below:



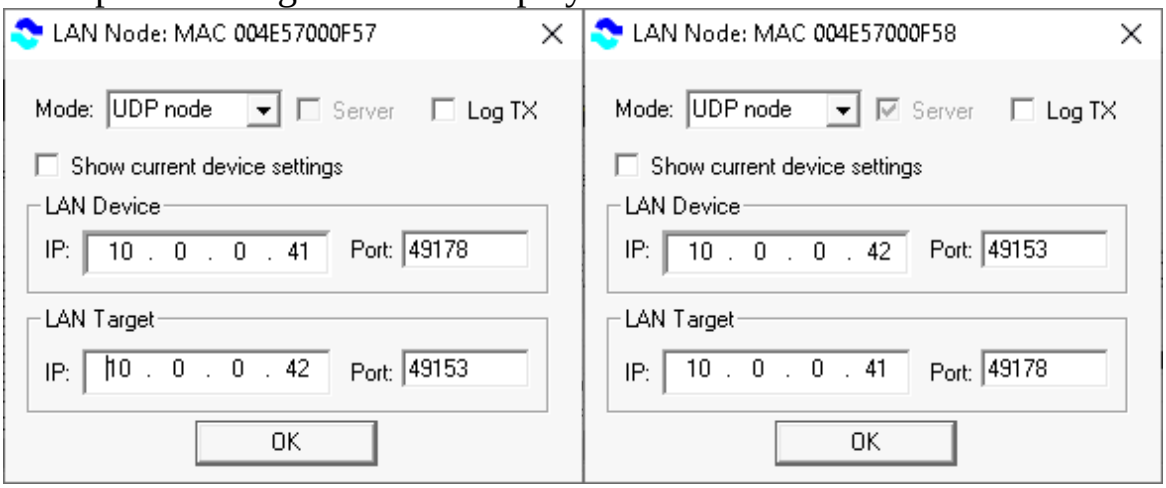
Note: Device IP of node 1 equals target IP of node 2.
Device IP of node 2 equals target IP of node 1.

Client node can start communications by performing the following command:
ENABLE enet; //connect to server
Client node stops communications by performing this: DISABLE enet;//disconnect from server
The connection status can be verified by doing this: SET x20 = enet;//1-connected, 2-ready to connect, 0-unable to connect due to node not being bonded to DHCP.
Alternatively, connection verification can be done as follows: IF enet = 1; //if connected
//perform some tasks;
ENDIF;
When client and server nodes are connected, they are able to communicate via LAN - TX and RX LAN messages.


Netway nw92 via LibUsb1 [SIM GM Eboost BEF TCP]
File View Edit Monitor Help
EMU
Time(h:m:s.ms) Type Dlen Filter Frame
6 22 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26
7 22 27 28 29 2A 2B 2C 2D 2C 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B
8 4 3C 3D 3E 3F
9 0:40:26.634 lan 16 userflt3 <10.0.0.42> 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
10 22 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26
11 22 27 28 29 2A 2B 2C 2D 2C 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B
12 4 3C 3D 3E 3F
SEND
CL 3
connect client to LAN server!
Run emu
All traffic
Log:[0-940]
Mon:[0-940]
Cmd
User Messages
Item Trigger Command Line Tag Description
1 SET timer0 = 0h; CL cl-tag3; SET timer1 = BB8h; SET timer31 = 7D0h... 1
2 SET x1 = 0h; SET x2 = 0h; SET xbit200 = 0h; SET x3 = 0h; SET x4 =... 3
3 ENABLE enet; connect client to LAN ser...
4 SET x20 = x22; SET x20 & Fh; IF x20 < Ah; SET x20 + 30h; ELSE; S... 10 Convert single digit Hex '0...
5 DISABLE enet; disconnect from server
6 QUE usermsg3;
7 timer1 SET timer1 = 1F4h; SET x20 = enet20; SET xbit40304 = 0h; IF xbit20... T_MainLoop
8 IF timer3 < 9470h; IF timer3 > 1F40h; SET x403 = 1h; SET x404 = 2h;...
Device Time: 0:40:28.420 Rx: 1194 Tx: 0 Log: 76 Warnings: 0 device:3927, fw:8.22b8

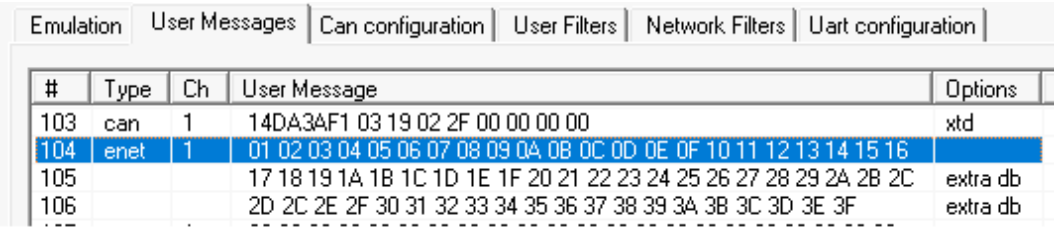
UDP mode:

When in UDP mode, devices operate without external DHCP server.
Example of configurations is displayed below:



Transmitting LAN messages

User can send enet message by using two methods:
1. QUE usermsg104; //this will send 64-byte message specified as shown below:

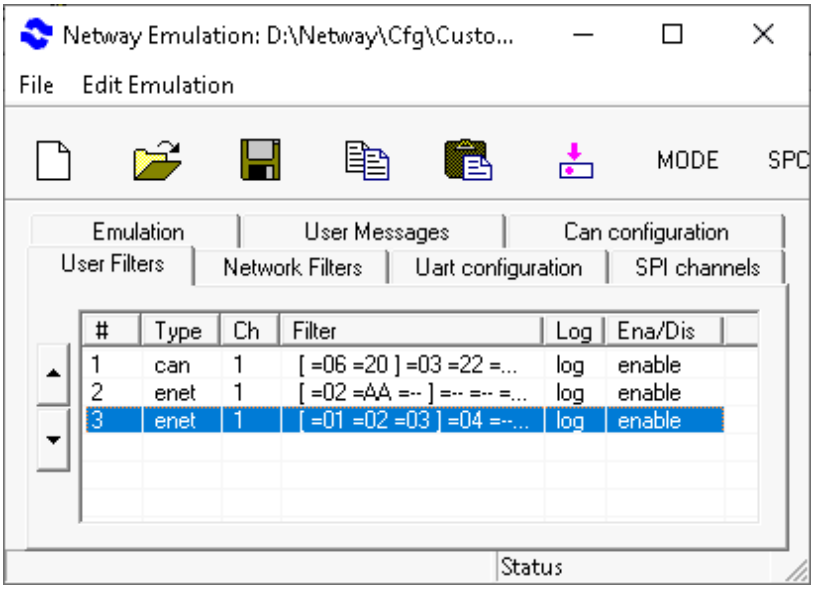

Emulation
User Messages
Can configuration
User Filters
Network Filters
Uart configuration
Type Ch User Message Options
103 can 1 14DA3AF1 03 19 02 2F 00 00 00 00 xtd
104 enet 1 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16
105 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C extra db
106 2D 2C 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F extra db

2. Set TxLong with enet protocol (22), assign data byte to X-VAR array and send it with QUE command:

```
IF enet = 1;//if connected
SET TxLong1 = protocol22; // enet
SET TxLong1 = bytesize64; // 64 Bytes of data
SET TxLong1 = x399; //1st byte to send starts at x399.
QUE TxLong1;
ENDIF;
```

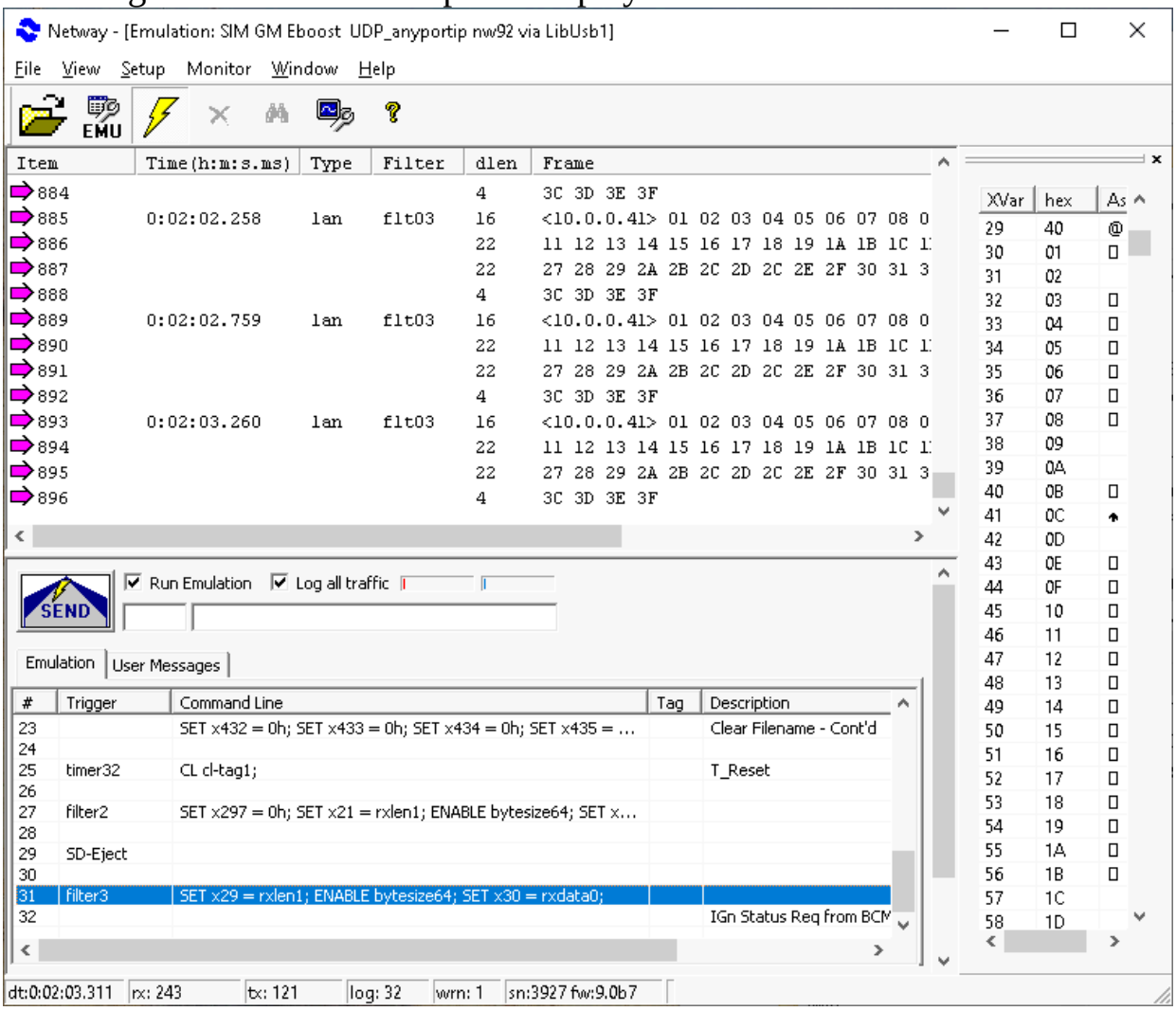
Receiving LAN messages

Connected device receives the enet message sent by another device (usermsg104). User can set user i.e. filter #3 if there is a need to process received data:



In emulation user can configure event triggered by filter3: filter3 --> SET x29 = rxlen1;//store number of RX data bytes in message ENABLE bytesize64;

SET x30 = rxdata0;//copy 64 bytes into x30-x93 array for further analysis
Resulting traffic and X-VAR panel displayed below:



Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

How Emulation works

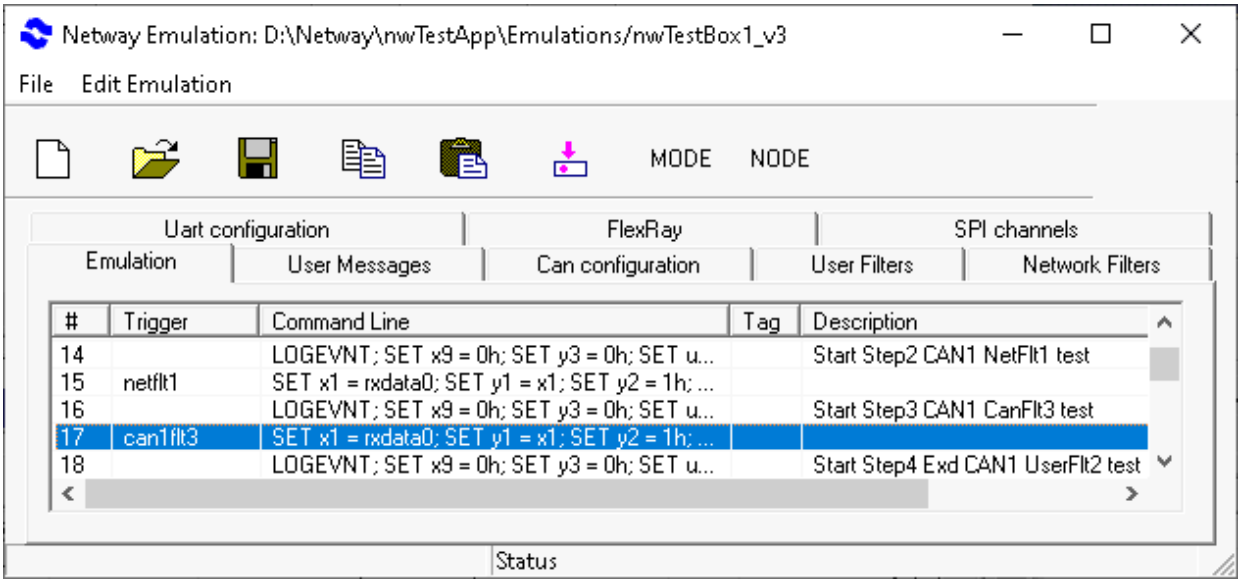
Using Netway application user creates emulation file including setup of all device resources, protocols handling, filtering, logging and reactions to external events.
Once file is downloaded into device it is verified for validity and saved into the flash memory, so on next power up it will define device's behavior until different emulation is downloaded.
All Netway events performed in a stand-alone mode, connection to PC is not necessary.
There are few major items defined in emulation:

1. **Events** - event trigger and command line determine when and how event is triggered and what it will do. Event triggers include incoming messages, timers, digital input transitions. Events can be "called" from other events as well.
2. **User messages** - user specifies outgoing messages which will be transmitted by command lines of an event.
3. **User filters / Network filters** - specifying incoming messages which can be used as event triggers and/or for logging purposes.
4. **CAN configuration** - specifies CAN network parameters and CAN filters which can be used as event triggers
5. **Flexray configuration** - specifies Flexray network parameters and objects. RX objects can be used as event triggers.
6. **UART configuration** - specifies UART parameters and protocol for each channel.
7. **SPI/I2C/J1850 configuration** - specifies parameters for selected network protocol

Next few topics describe the above items in more details.

Emulation Events

Emulation events shown below allow to add / delete / edit all three columns: *Trigger*, *Command Line* and *Description*.
Right-click on selected event displays operation menu items, double-click displays edit of selected item.


Netway Emulation: D:\Netway\nwTestApp\Emulations/nwTestBox1_v3
File
Edit Emulation
MODE
NODE
Uart configuration
FlexRay
SPI channels
Emulation
User Messages
Can configuration
User Filters
Network Filters
Trigger Command Line Tag Description
14 LOGEVNT; SET x9 = 0h; SET y3 = 0h; SET u... Start Step2 CAN1 NetFlt1 test
15 netflt1 SET x1 = rxdata0; SET y1 = x1; SET y2 = 1h; ...
16 LOGEVNT; SET x9 = 0h; SET y3 = 0h; SET u... Start Step3 CAN1 CanFlt3 test
17 can1flt3 SET x1 = rxdata0; SET y1 = x1; SET y2 = 1h; ...
18 LOGEVNT; SET x9 = 0h; SET y3 = 0h; SET u... Start Step4 Exd CAN1 UserFlt2 test
Status

Mode:

Operating mode:

Controlled by user:

- **Run Emulation** - when unchecked emulation is disabled
- **Run CL1 on power-up** - command line1 will run after power-up
- **Power-down mode** - when checked and traffic inactivity detected for 120 second,
  device switches OFF. When traffic resumes device switches ON.

Read only:

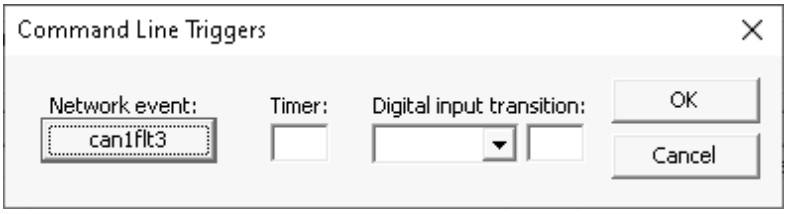
- **EMU Checksum OK** - indication of emulation status
- **Real-Time Clock OK** - RTC OK status
- **MsgBuffer over-run** - indication of message buffer overrun.
- **Device enabled** - when checked, security is OK, emulation and log enabled

Log mode:

- **Enable log** - when checked
- **Log all network traffic** - log all not specified by filters when checked
- **Protocol violations** - log when checked

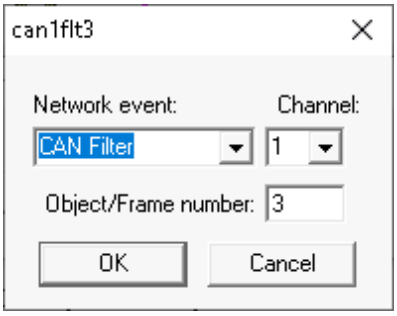
- **IFR violations** - log when checked

Triggers:



There are three types of Command line trigger: *Network Event*, *Timer*, *Digital Input Transition*

Network Event:



Selection of Network events:

- CAN Filter - object is equal filter number from CAN Configuration page of selected channel
- Flexray Frame - frame number from Flexray Configuration page of selected channel
- LIN Frame - frame number from Uart Configuration page of selected channel (if LIN protocol is selected)
- Network Filter - filter number from Network Filters page
- User Filter - filter number from User Filters page
- MultiFrameRX CAN - each CAN channel can have up to 4 multi-frame channels assigned.
- MultiFrameTX CAN - see previous item
- Uart LONG - long Uart message RX event
- J1708 PID - triggers upon detecting specified PID number
- Warning - triggers when specific warning is detected
- J1939 Diagnostics

- J1708 Diagnostics

Timer:

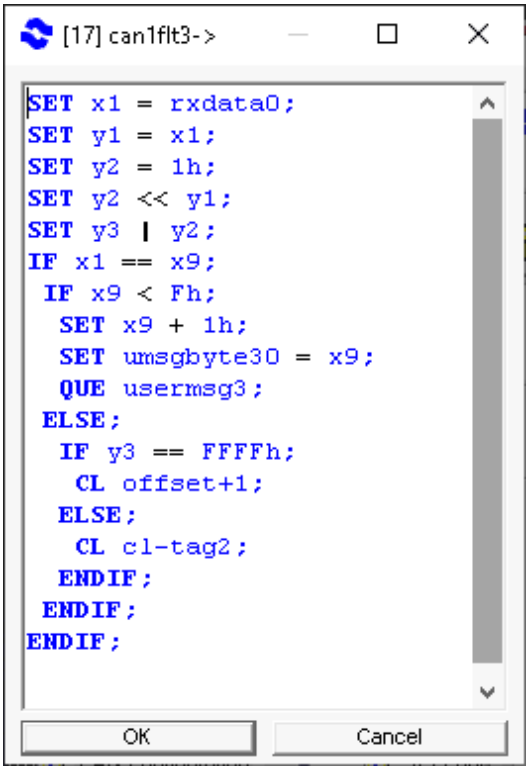
Specify timer number (1 -:- 32). when timer expires the command line will be triggered

Digital Input Transition:

Following selections are available:
= 0 - if digital input number is less than 3V, command line allowed to be performed
= 1 - if digital input number is greater than 3V, command line allowed to be performed
0->1 - positive signal transition will trigger command line
1->0 - negative signal transition will trigger command line

Command line:

Double click on Command line column to bring forward edit dialog to allow modify commands for this event, see example below:



Note: Max 24 commands per command line

Tag:

Unique number to identify event/command line allows to reference this event from another command line.
Double-click to modify.

Description:

After double-clicking on description column user can type in description for this event

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

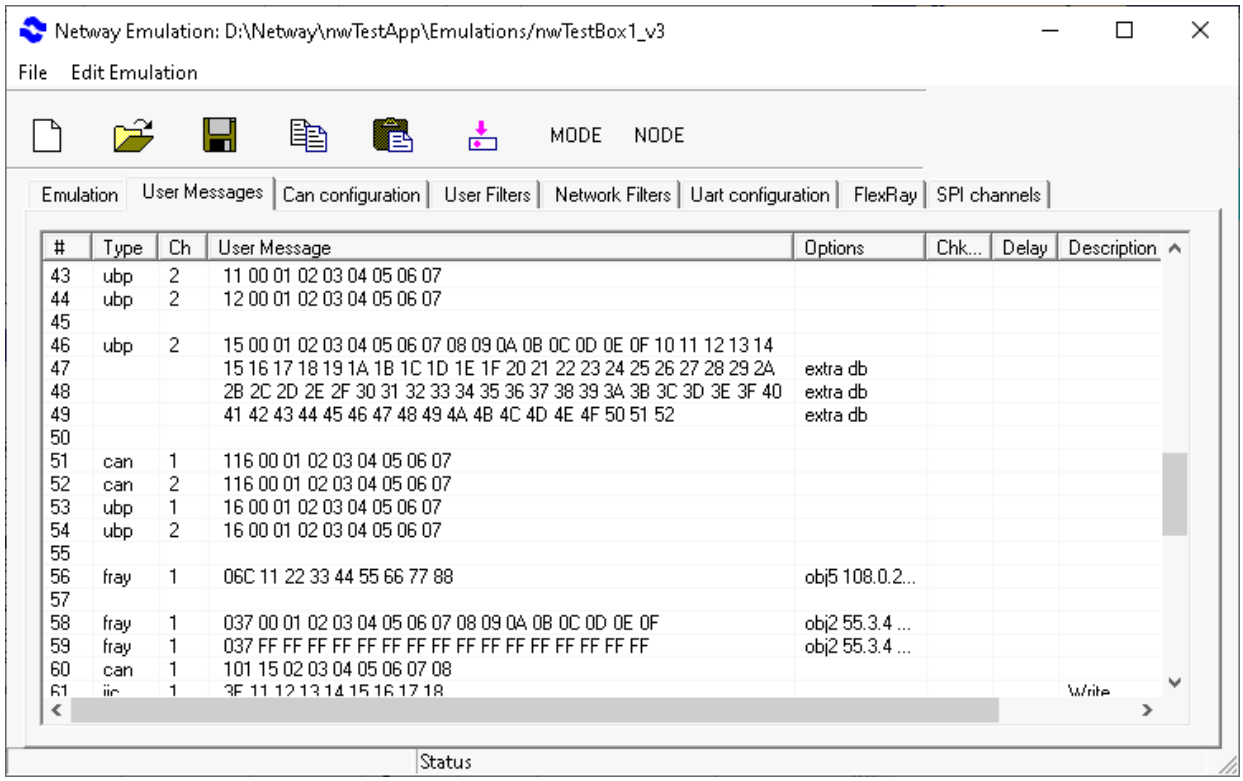
User Messages

Up to 255 user messages can be configured for transmitting by the Netway device.

Eight columns are shown below:

- **#** - user message number, this will be used to reference user message in emulation script
- **Type** - user message protocol / type
- **Ch** - user message channel
- **User message** - message id and data bytes
- **Options** - specific options selection
- **Checksum** - for selected protocols allows to specify checksum (including invalid)
- **Delay** - for selected protocols allows to specify delay
- **Description**

Double-click on selected message column allows user to modify message type/content/options/description
Note: during emulation run user message content can be changed by emulation commands as well.


Netway Emulation: D:\Netway\nwTestApp\Emulations/nwTestBox1_v3
File
Edit Emulation
MODE
NODE
Emulation
User Messages
Can configuration
User Filters
Network Filters
Uart configuration
FlexRay
SPI channels
Type Ch User Message Options Chk... Delay Description
43 ubp 2 11 00 01 02 03 04 05 06 07
44 ubp 2 12 00 01 02 03 04 05 06 07
45
46 ubp 2 15 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14
47 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A extra db
48 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 extra db
49 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 extra db
50
51 can 1 116 00 01 02 03 04 05 06 07
52 can 2 116 00 01 02 03 04 05 06 07
53 ubp 1 16 00 01 02 03 04 05 06 07
54 ubp 2 16 00 01 02 03 04 05 06 07
55
56 fray 1 06C 11 22 33 44 55 66 77 88 obj5 108.0.2...
57
58 fray 1 037 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F obj2 55.3.4 ...
59 fray 1 037 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF obj2 55.3.4 ...
60 can 1 101 15 02 03 04 05 06 07 08
61 iic 1 3F 11 12 13 14 15 16 17 18 Write
Status

User Message Types

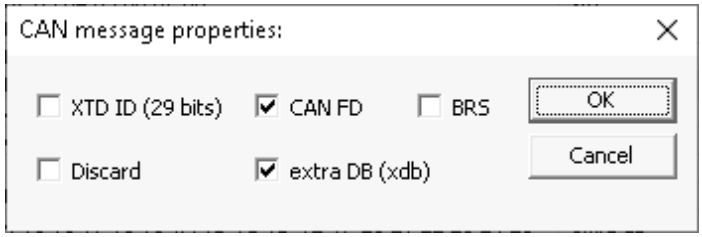
- **scp** - J1850 Ford
- **cl2** - J1850 GM
- **can** - CAN (FD) protocol
- **ubp** - Uart Based Protocol
- **i2c** - I2C protocol
- **spi** - SPI protocol
- **fray** - Flexray
- **enet** - Ethernet LAN
- **sent** - single edge transition protocol

User Message Body

Double-click to modify ID / Data Bytes

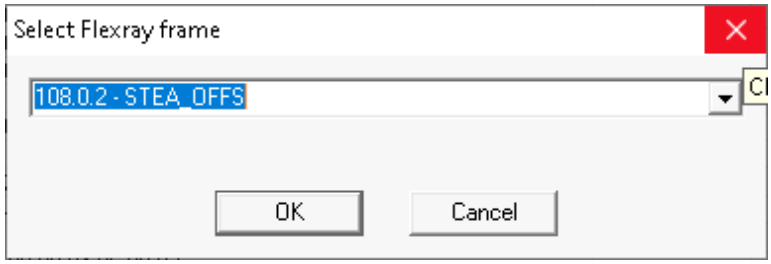
Options

Options dialog for CAN message:



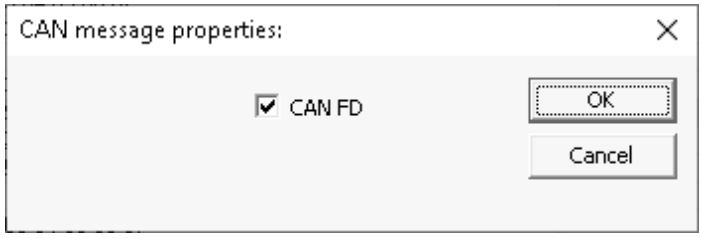
- **XTD ID** - extended CAN ID 29-bit
- **CAN FD** - flexible data
- **BRS** - bit-rate switch (data bytes bit-rate higher then ID rate)
- **Discard** - discard log
- **extra DB** - data payload above 8 first data bytes

Options dialog for Flexray message



When Flexray object ID matches the spec, the selection is available as shown above

Options dialog for none protocol message

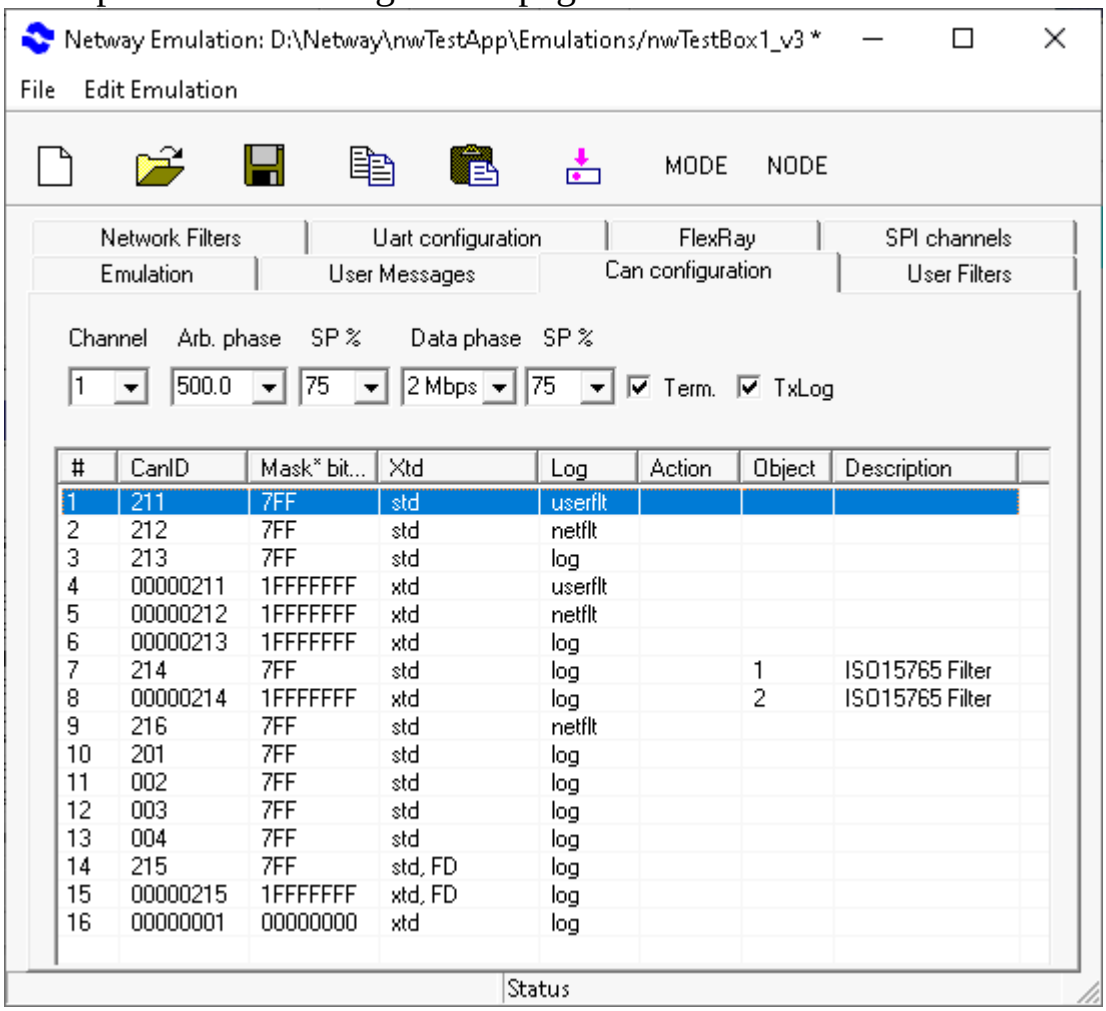


When checked extra data bytes can be specified to continue for the previous user message

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

CAN Configuration

Example of CAN configuration page for channel 1 is shown below:



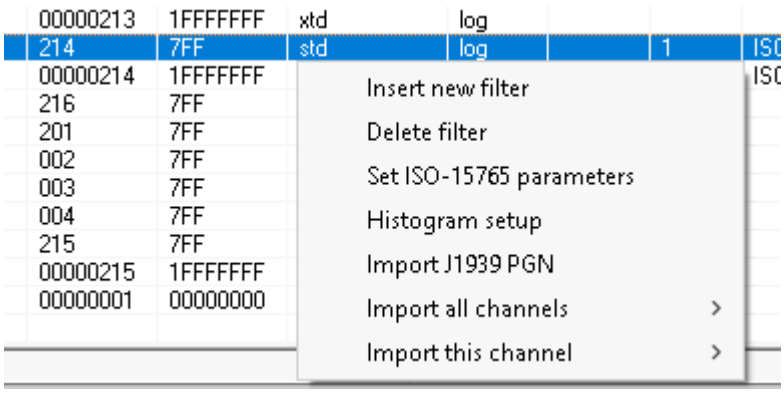
- **Channel** number - 1-:-8
- **Arb. phase** - arbitration (ID) phase bit-rate
- **SP** - sample point (%) for ID
- **Data phase** - bit-rate for data (FD only)
- **SP** - sample point (%) for data bytes
- **Term** - termination resistor ON/OFF
- **TxLog** - log/discard transmitted messages for this channel

CAN filters list

- # - filter number used as a reference in emulation
- **Can ID**
- **Mask** - bit mask, 0-do not care, 1-match, can1flt16 will work for any ID not intercepted by previous filters.
- **Xtd** - extended-29bit/standard-11bit ID
- **Log** - how to process: Log/Discard/Netflt/Userflt
- **Action** - when match: do nothing / TX user message number / gateway to channel number
- **Object** - number (channel or user message) for the action

List Menu commands

Right-click exposes available menu commands:

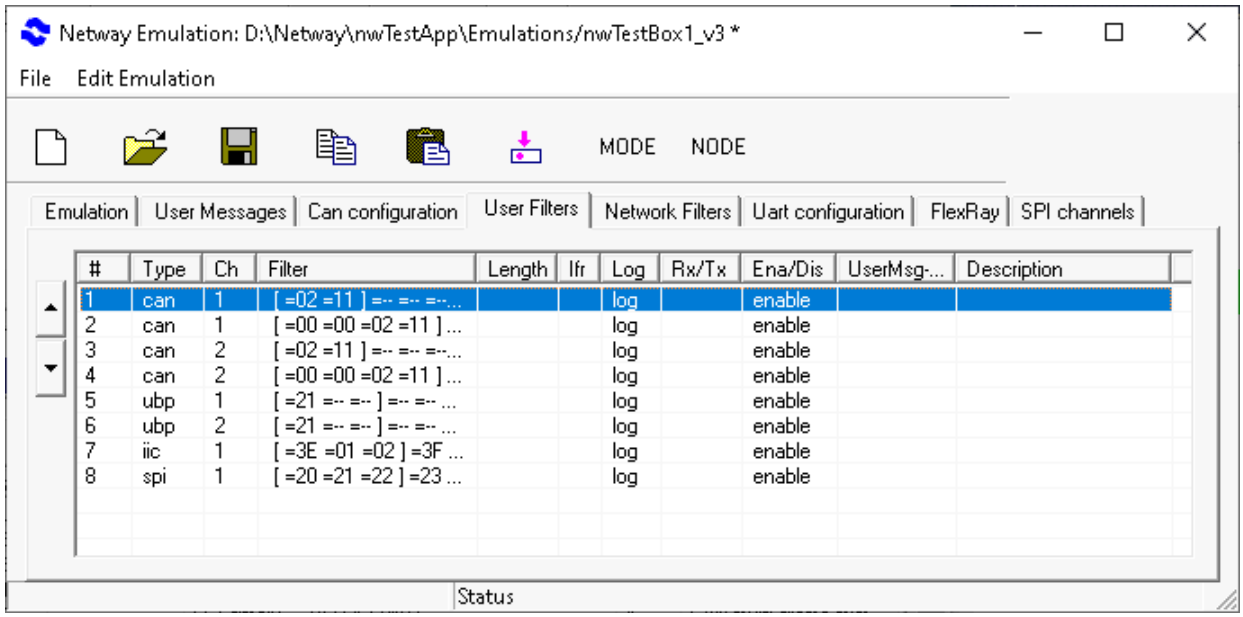


- Insert / Delete filter
- Set multi-frame parameters
- Import

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

User Filters

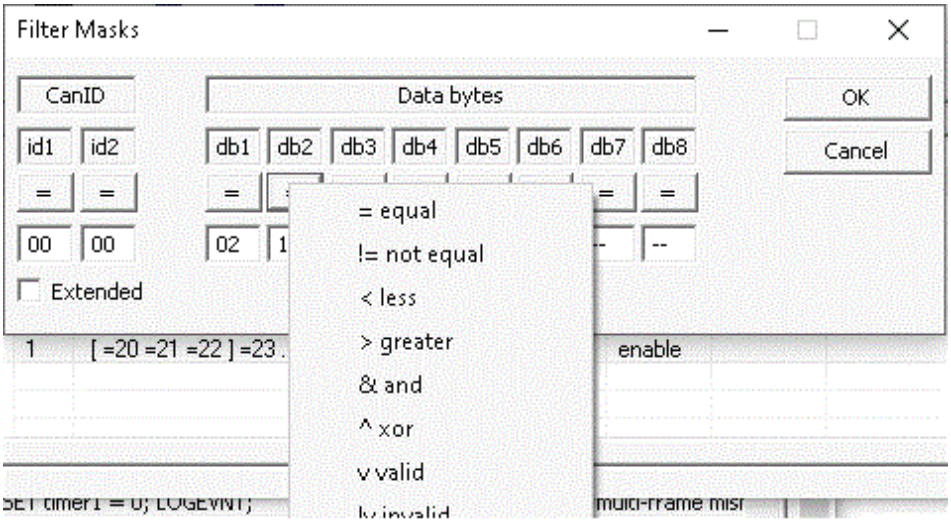
Example of User Filters is shown below:


Netway Emulation: D:\Netway\nwTestApp\Emulations/nwTestBox1_v3 *
File
Edit Emulation
MODE
NODE
Emulation
User Messages
Can configuration
User Filters
Network Filters
Uart configuration
FlexRay
SPI channels
Type Ch Filter Length Ifr Log Rx/Tx Ena/Dis UserMsg-... Description
1 can 1 [=02 =11] =-- =-- =--... log enable
2 can 1 [=00 =00 =02 =11] ... log enable
3 can 2 [=02 =11] =-- =-- =--... log enable
4 can 2 [=00 =00 =02 =11] ... log enable
5 ubp 1 [=21 =-- =--] =-- =-- ... log enable
6 ubp 2 [=21 =-- =--] =-- =-- ... log enable
7 iic 1 [=3E =01 =02] =3F ... log enable
8 spi 1 [=20 =21 =22] =23 ... log enable
Status

User Filter allows to compare/match incoming message in most versatile way.

- **#** - filter number, used as a reference in emulation
- **Type** - filter protocol type
- **Ch** - protocol channel number
- **Filter** - compare/match incoming message to up to 11 first bytes
- **Ifr** - in-frame response match
- **Log** - log/discard
- **Rx/Tx** - TX/TX or both
- **Ena/Dis** - initial enable/disable status
- **UserMsg** - rapid response with user message number
- **Description** - optional command line description

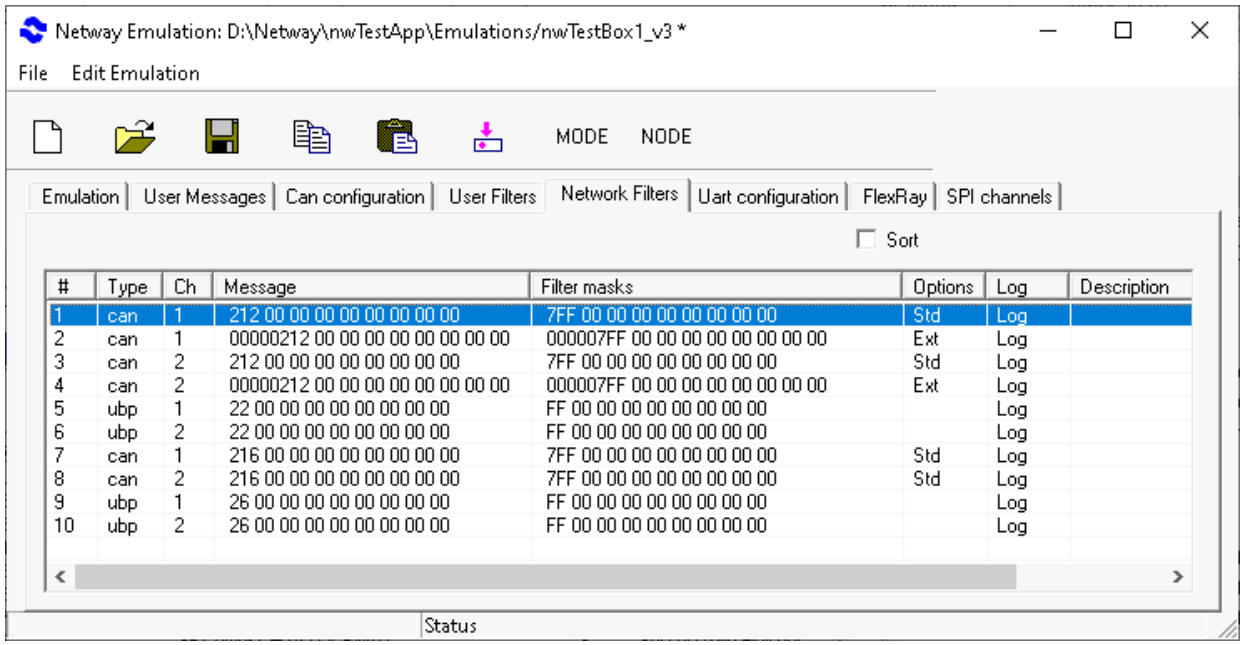
After double-click on filter columns 11 data fields can be specified. Use '--' for do not match this element.



Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

Network Filters

Example of Network filters is shown below:

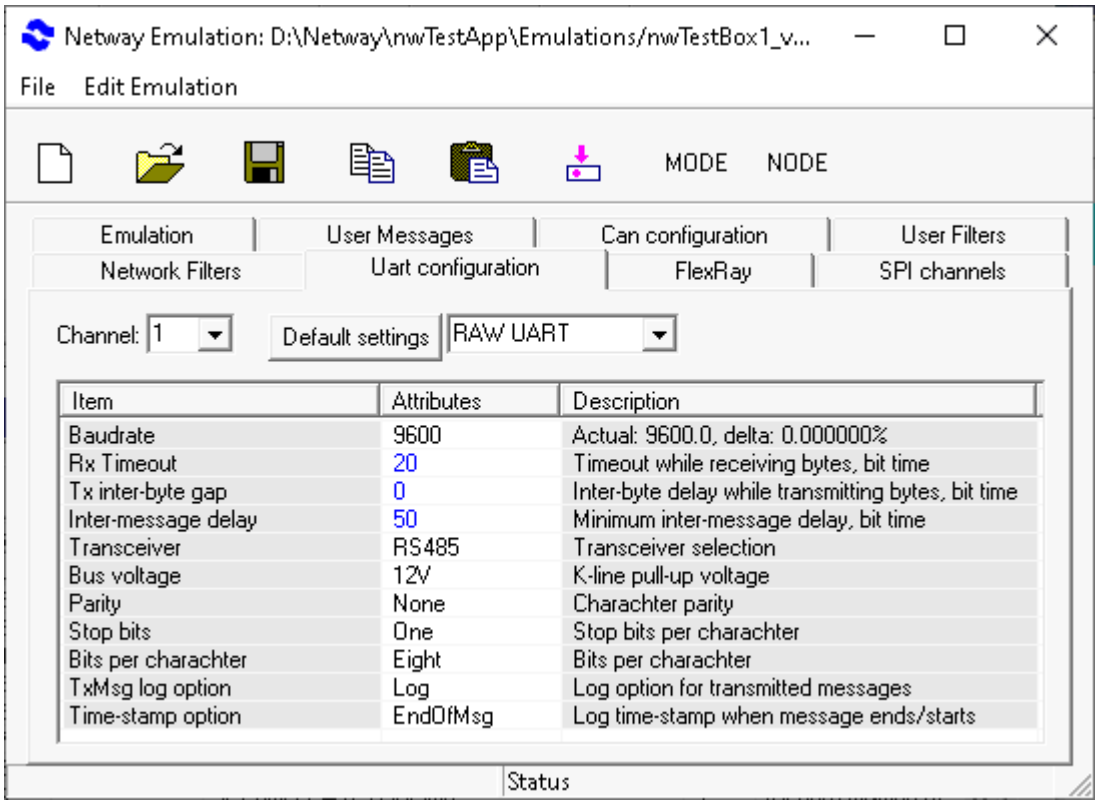


Double-click to modify applicable columns

- # - filter number used as a reference in emulation
- **Type** - protocol type
- **Ch** - protocol channel
- **Message** - data bytes to match
- **Filter masks** - bit masks, 0-do not care, 1- match
- **Options** - Std/Exd for CAN messages
- **Log** - Log/Discard
- **Description** - optional command line description

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

Uart Configuration



- **Channel** - Uart channel number (1-:-8)
- **Uart protocol** selection - UBP(Ford), ISO9141(Ford), ACP(Ford), KWP2000, LIN, ISO9141-2, RAW, J1708, RS232, Profibus
- **Baudrate** - bit-rate selection
- **Rx Timeout** - end of frame when specified time is expired
- **Tx inter-byte gap** - minimum time between TX messages
- **Transceiver** - choose between LIN / RS485 / RS232 / TTL transceivers
- **Bus voltage** - 12V/5V selection
- **Parity** - none / odd / even
- **Stop bits** - 1 or 2
- **Bits per character** - 5, 6, 7, 8, 9
- **TxMsg log option** - log/discard
- **Time-stamp option** - start / end of message

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

Syntax

Netway Script language allows to perform commands / operations between objects - operand1 and operand2:
CMD <operand1> <operator> <opeand2>; //the angle brackets signify optional content
Objects are usually followed by numeric values signifying object number.
All Netway statements are case sensitive, objects and operator separated by space.
Complete command terminated by semicolon.
Netway device operates with objects such as: messages, bytes, bits, filters, variables, timers, etc.
Therefore we are going to use terms *object* and *operand* interchangeably
Maximum 24 commands per command line/event. Maximum 196 command lines / events per emulation.
When typing command line, the syntax is verified, text turns red when not valid.
When unsure of syntax, highlight the object and clarifying hint will be displayed:

Commands:

Limited number of commands is allowed: SET - set operand1 to operand2 (SET x1 = 201;) QUE - queue user message or event (QUE usermsg1;) ENABLE - enable resource (ENABLE filter1;) DISABLE - disable resource (DISABLE timer1;) IF - condition (IF x1 = rxbyte3;) ENDIF - end of condition (ENDIF;) ENDCL - end of command line (ENDCL;) CL - perform referenced command line, event (CL 45; CL cl-tag3; CL offset+1;)

LOGEVNT - log event special message (LOGEVNT;) LOG - log variables in special message (LOG x1;)

RESETTMR - reset device time stamp (RESETTMR;) NETMODE - set special network mode (NETMODE 51;) ELSE - condition opposite to IF (ELSE;) ELSEIF - condition continues (ELSEIF x1 = rxbyte4;) *Note*: IF command should have matching ENDIF;

Operators:

= equal, + plus , - minus, * multiply , / divide, | or , & and, ^ xor , ~ not, >> shift right , << shift left

Operands / Objects:

Variables:

- **x** - 1024 8-bits variables, x1 -:- x1024, *Example*: SET x45 = 234; SET x46 = rxdata3; SET x47 = F1h; IF x48 = 32;
- **y** - 256 16-bits variables, y1 -:- y256
- **z** - 64 32-bits variables, z1 -:- z64
- **xbit**NUMBB - bit of x-variable NUM, BB = 0-:-7; *Example*: x45, bit7: SET xbit4507 = 1; IF xbit4507 = 1;
- **ybit**NUMBB - bit of y-variable NUM, BB = 0-:-15; *Example*: y45, bit11: SET ybit4511 = 1; IF ybit4511 = 1;
- **zbit**NUMBB - bit of z-variable NUM, BB = 0-:-31; *Example*: z45, bit0: SET zbit4500 = 1; IF zbit4500 = 1;
- **xy**NUM - x-variable referenced by y-variable NUM; *Example*: SET xy100 = 55h;
- **indX**NUM - x-variable referenced by x-variable NUM
- **indY**NUM - y-variable referenced by y-variable NUM
- **block**OFS - 8-bit variable in block memory referenced by offset

User Message:

- **usermsg** - user message followed by number. *Example*: QUE usermsg24;//transmit
- **umsgbyte** - user message byte. *Example*: SET umsgbyte2502 = 45h;//user message25, byte 2 = 45 hex
- **umsgbit** - - user message bit. *Example*: IF umsgbit2543 = 1;//if user message25, byte4, bit3
- **umsgword** - user message word. *Example*: SET umsgword252 = 2301h;//user message25 byte2=23h, byte3=01h;
- **umsg**X - user message referenced by xNUM. *Example*: QUE umsgX2;//TX user message referenced by X2

Receive Message/Frame:

rxbyte - RX byte including ID, index referenced. *Example*: SET x10 = rxbyte0;//copy all message bytes to x10, x11...

SET x20 = rxbyte2;//copy single byte to x20
ENABLE bytesize5; SET x30 = rxbyte3;//copy 5 bytes (3,4,5,6,7) to x30-x34

rxdata - RX data byte, excluding ID, index referenced. *Example*: SET x10 = rxdata0;//copy first byte bytes to x10
SET x20 = rxdata2;//copy third byte to x20
ENABLE bytesize5; SET x30 = rxdata3;//copy 5 bytes (4,5,6,7,8) to x30-x34

rxbit - binary value of RX byte N, bit M, M range: 0-:-7. *Example*: IF rxbit14 = 1;//if byte1, bit4 equals 1
SET x1 = rxbit107;//byte10, bit7 to x1

rxlen - Message/frame length in bytes IF rxlen1 = 8;//if length of RX message equals 8

RxLong - content of RX long message. *Example*: SET x1 = RxLong1;//Copy all RX bytes to x1,x2...

RxLongLen - Long Message length in bytes. *Example*: SET x2 = RxLongLen1; //Length of received long message to x2

Filters / Triggers:

- **filter** - user filter
- **netflt** - network filter
- **can**N**flt**NUM - can filter NUM of channel N
- **frayFrame** - Flexray object / frame
- **linFrame** - LIN frame

IO

- **analog**N - analog input / output N in mV. *Example*: SET analog2 = 1000;//set analog output; SET y1 = analog3;//analog input 3 to y1;
- **diginp**N - digital input N. *Example*: IF diginp1 = 1;
- **digout**N - digital output N. *Example*: SET digout2 = 1;
- **pwmFreq**N - output PWM frequency N. *Example*: SET pwmFreq4 = 1000;//in kHz
- **pwmDuty**N - output PWM duty cycle N. *Example*: SET pwmDuty4 = 20;//in %
- **inFreq**N - input PWM frequency N. *Example*: SET y1 = inFreq4;//in kHz

- **inDuty**N - input PWM duty N. *Example*: SET x1 = inDuty4;//in %
- **led**N - output LED N. *Example*: SET led3 = 1;//turn ON, SET led3 = 0;//turn OFF

Usage of Netway script language is demonstrated in Example Emulation

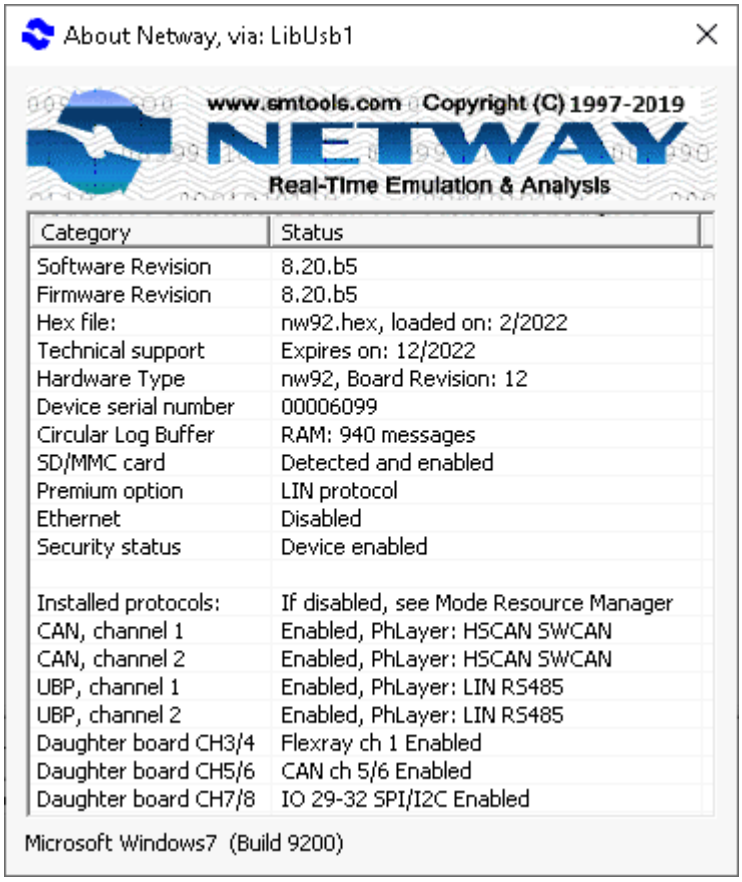
Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

Example Emulation

To demonstrate Netway device functionality in this manual we are going to use two devices configured with nwTestBox1 and nwTestBox2 emulations. Box1 (nwTestBox1.emu) will be initializing communications via different networks (protocols) by sending messages to Box2 (nwTestBox2.emu). Box2 will be sending responses to Box1. Based on response bytes/bits Box1 will declare test step pass/failed and continue to the next step if pass. For this emulation the devices under test have following networks enabled:

- CAN channel 1
- CAN channel 2
- Uart channel 1
- Uart channel 2
- SPI/I2C/J1850 channel 1
- CAN channel 5,6 channel 2
- Flexray channel 3

About Netway dialog shows those networks below:


About Netway, via: LibUsb1
www.smtools.com Copyright (C) 1997-2019
NETWAY
Real-Time Emulation & Analysis
Category
Status
Software Revision
8.20.b5
Firmware Revision
8.20.b5
Hex file:
nw92.hex, loaded on: 2/2022
Technical support
Expires on: 12/2022
Hardware Type
nw92, Board Revision: 12
Device serial number
00006099
Circular Log Buffer
RAM: 940 messages
SD/MMC card
Detected and enabled
Premium option
LIN protocol
Ethernet
Disabled
Security status
Device enabled
Installed protocols:
If disabled, see Mode Resource Manager
CAN, channel 1
Enabled, PhLayer: HSCAN SWCAN
CAN, channel 2
Enabled, PhLayer: HSCAN SWCAN
UBP, channel 1
Enabled, PhLayer: LIN RS485
UBP, channel 2
Enabled, PhLayer: LIN RS485
Daughter board CH3/4
Flexray ch 1 Enabled
Daughter board CH5/6
CAN ch 5/6 Enabled
Daughter board CH7/8
IO 29-32 SPI/I2C Enabled
Microsoft Windows7 (Build 9200)

Initialization

Command line 1 is run after power-up. In our case we use this event to initialize variables and kick-off timer3:

```
SET x0 = 0h;
SET y0 = 0h;
SET timer3 = 1000; //1 second delay
```

Event 76 is triggered by timer3 ->

```
SET z1 = diginp0; //32 digital inputs to z1
SET timer3 = 500; //do it every 500 ms.
```

Test Failed Events:

Event2: timer1 -> //Time-out - Test failed!
LOGEVNT; //log event trace
SET digout14 = 0;

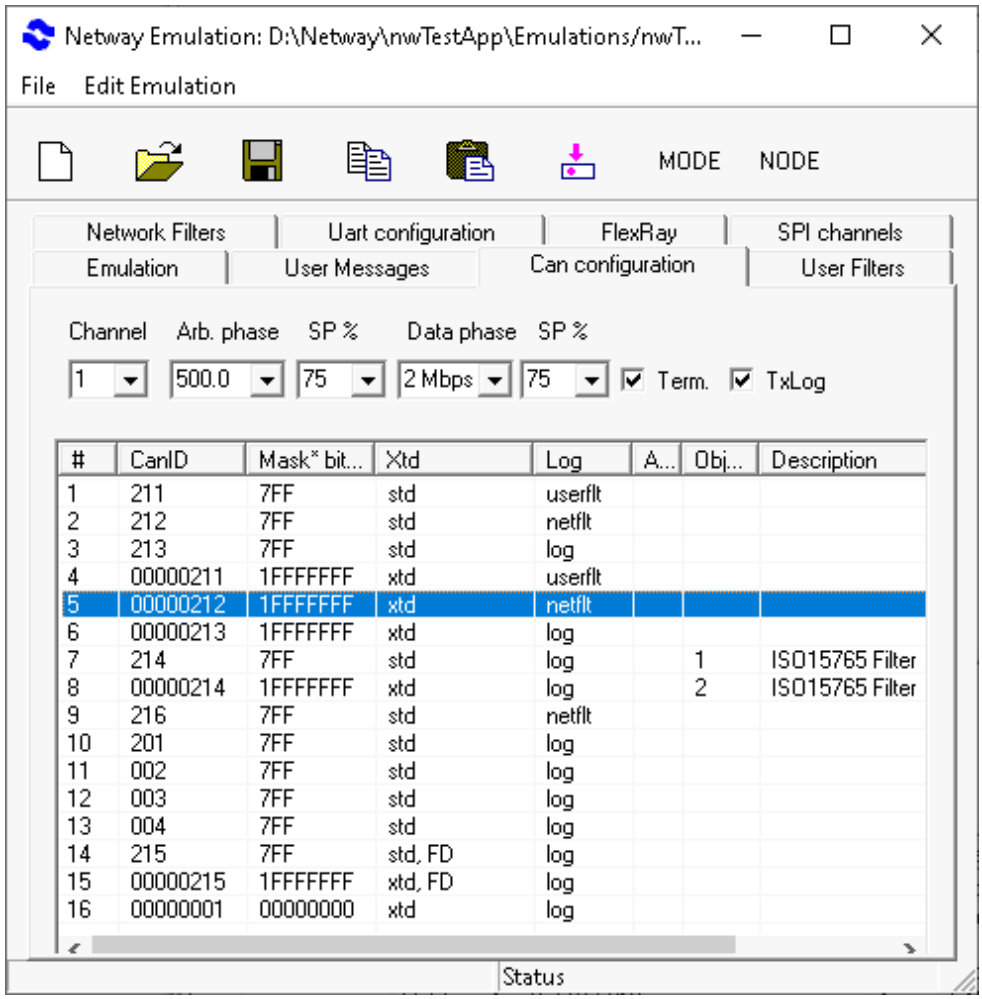
Event 3,4,5,6,7,8,9,10,11
SET timer1 = 0;//turn off timer1
LOGEVNT;//log event trace

Test Steps:

- CAN1 Test
- CAN2 Test
- Uart1 Test
- CAN1/2 + Uart1/2 Test
- Flexray Test
- SPI / I2C Test
- CAN5/6 Test
- J1708 / J1939 Test

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

CAN1 Test Steps



#	CanID	Mask* bit...	Xtd	Log	A...	Obj...	Description
1	211	7FF	std	userflt			
2	212	7FF	std	netflt			
3	213	7FF	std	log			
4	00000211	1FFFFFFF	xtd	userflt			
5	00000212	1FFFFFFF	xtd	netflt			
6	00000213	1FFFFFFF	xtd	log			
7	214	7FF	std	log		1	ISO15765 Filter
8	00000214	1FFFFFFF	xtd	log		2	ISO15765 Filter
9	216	7FF	std	netflt			
10	201	7FF	std	log			
11	002	7FF	std	log			
12	003	7FF	std	log			
13	004	7FF	std	log			
14	215	7FF	std, FD	log			
15	00000215	1FFFFFFF	xtd, FD	log			
16	00000001	00000000	xtd	log			

CAN configuration channel 1 is shown above.

Step 1: CAN1 UserFlt1 Test

Box1 user message1: can1->111 00 00 00 00 00 00 00 00
Box1 user filter1: can1->211 -- -- -- -- -- -- -- --
Box1 sends can1->111 NN 00 00 00 00 00 00 00
Box2 responds with can1->211 NN 00 00 00 00 00 00 00 (flt01) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event12: Start Step1

```
IF protocol3 == 0h; //if CAN1 present
CL cl-tag6; //Event 7 test failed
ENDCL; //end of command line
ENDIF; //end of condition
LOGEVNT; //log event trace
SET x1 = 0h; //initialize count
SET x9 = 0h; //initialize x9 with count
SET umsgbyte10 = x9; //set user message byte 0 to x9
QUE usermsg1; //TX user message 1
SET y3 = 0h; //initialize RX flags
SET timer1 = 10000; //allow 10 seconds for test
```

Event13: filter1->

```
SET x1 = rxdata0; //first data byte from RX message to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set y3 with flag corresponding to RX byte IF x1 == x9; //if RX byte equal TX byte
IF x9 < Fh; //if less than 16 messages has been received SET x9 + 1h; //increment TX byte SET umsgbyte10 = x9; //set first byte of user message1 to x9
    QUE usermsg1; //TX user message1
ELSE; //end of test
    IF y3 == FFFFh; //all messages have been received?
    CL offset+1; //go to step 2
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 1 Log:

Item	Time(h:m:s.ms)	Type	Filter	d..	Frame	Description
1	3:57:31.838	ind			Event12 RTC: 14:01:25: D...	Start Step1 CAN...
2	3:57:31.838	canl	TxCanl	8	111 00 00 00 00 00 00 00 00	
3	3:57:31.839	canl	flt01	8	211 00 00 00 00 00 00 00 00	
4	3:57:31.839	canl	TxCanl	8	111 01 00 00 00 00 00 00 00	
5	3:57:31.840	canl	flt01	8	211 01 00 00 00 00 00 00 00	
6	3:57:31.841	canl	TxCanl	8	111 02 00 00 00 00 00 00 00	
7	3:57:31.842	canl	flt01	8	211 02 00 00 00 00 00 00 00	
8	3:57:31.842	canl	TxCanl	8	111 03 00 00 00 00 00 00 00	
9	3:57:31.844	canl	flt01	8	211 03 00 00 00 00 00 00 00	
10	3:57:31.845	canl	TxCanl	8	111 04 00 00 00 00 00 00 00	
11	3:57:31.845	canl	flt01	8	211 04 00 00 00 00 00 00 00	
12	3:57:31.846	canl	TxCanl	8	111 05 00 00 00 00 00 00 00	
13	3:57:31.847	canl	flt01	8	211 05 00 00 00 00 00 00 00	
14	3:57:31.848	canl	TxCanl	8	111 06 00 00 00 00 00 00 00	
15	3:57:31.849	canl	flt01	8	211 06 00 00 00 00 00 00 00	
16	3:57:31.850	canl	TxCanl	8	111 07 00 00 00 00 00 00 00	
17	3:57:31.851	canl	flt01	8	211 07 00 00 00 00 00 00 00	
18	3:57:31.851	canl	TxCanl	8	111 08 00 00 00 00 00 00 00	
19	3:57:31.852	canl	flt01	8	211 08 00 00 00 00 00 00 00	
20	3:57:31.853	canl	TxCanl	8	111 09 00 00 00 00 00 00 00	
21	3:57:31.853	canl	flt01	8	211 09 00 00 00 00 00 00 00	
22	3:57:31.854	canl	TxCanl	8	111 0A 00 00 00 00 00 00 00	
23	3:57:31.855	canl	flt01	8	211 0A 00 00 00 00 00 00 00	
24	3:57:31.856	canl	TxCanl	8	111 0B 00 00 00 00 00 00 00	
25	3:57:31.856	canl	flt01	8	211 0B 00 00 00 00 00 00 00	
26	3:57:31.858	canl	TxCanl	8	111 0C 00 00 00 00 00 00 00	
27	3:57:31.858	canl	flt01	8	211 0C 00 00 00 00 00 00 00	
28	3:57:31.859	canl	TxCanl	8	111 0D 00 00 00 00 00 00 00	
29	3:57:31.860	canl	flt01	8	211 0D 00 00 00 00 00 00 00	
30	3:57:31.860	canl	TxCanl	8	111 0E 00 00 00 00 00 00 00	
31	3:57:31.861	canl	flt01	8	211 0E 00 00 00 00 00 00 00	
32	3:57:31.862	canl	TxCanl	8	111 0F 00 00 00 00 00 00 00	
33	3:57:31.863	canl	flt01	8	211 0F 00 00 00 00 00 00 00	
34	3:57:31.863	ind			Event14 RTC: 14:01:25: D...	Start Step2 CAN...

Step 2: CAN1 NetFlt1 Test

Box1 user message2: can1->112 00 00 00 00 00 00 00 00
Box1 network filter1: can1->212 -- -- -- -- -- -- -- --
Box1 sends can1->112 NN 00 00 00 00 00 00 00
Box2 responds with can1->212 NN 00 00 00 00 00 00 00 (netflt01) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event14: Start Step 2

```
LOGEVNT; // log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //initialize x9 with count
SET umsgbyte20 = x9; //set user message2 byte 0 to x9
QUE usermsg2; //TX user message2
SET timer1 = 100; //allow100 ms for test
```

Event15: netflt1 ->

```
SET x1 = rxdata0; //first data byte from RX message to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set y3 with flag corresponding to RX byte IF x1 == x9; //if RX byte equal TX byte
IF x9 < Fh; //if less than 16 messages has been received SET x9 + 1h; //increment TX byte SET umsgbyte20 = x9; //set first byte of user message2 to x9
    QUE usermsg2; //TX user message2
ELSE; //end of test
    IF y3 == FFFFh; //all messages have been received?
    CL offset+1; //go to step 3
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step2 Log:

Item	Time(h:m:s.ms)	Type	Filter	d..	Frame	Description
34	3:57:31.863	ind			Event14 RTC: 14:01:25: D...	Start Step2 CAN...
35	3:57:31.864	canl	TxCanl	8	112 00 00 00 00 00 00 00 00	
36	3:57:31.865	canl	netfltl	8	212 00 00 00 00 00 00 00 00	
37	3:57:31.865	canl	TxCanl	8	112 01 00 00 00 00 00 00 00	
38	3:57:31.866	canl	netfltl	8	212 01 00 00 00 00 00 00 00	
39	3:57:31.867	canl	TxCanl	8	112 02 00 00 00 00 00 00 00	
40	3:57:31.868	canl	netfltl	8	212 02 00 00 00 00 00 00 00	
41	3:57:31.868	canl	TxCanl	8	112 03 00 00 00 00 00 00 00	
42	3:57:31.869	canl	netfltl	8	212 03 00 00 00 00 00 00 00	
43	3:57:31.870	canl	TxCanl	8	112 04 00 00 00 00 00 00 00	
44	3:57:31.870	canl	netfltl	8	212 04 00 00 00 00 00 00 00	
45	3:57:31.871	canl	TxCanl	8	112 05 00 00 00 00 00 00 00	
46	3:57:31.872	canl	netfltl	8	212 05 00 00 00 00 00 00 00	
47	3:57:31.873	canl	TxCanl	8	112 06 00 00 00 00 00 00 00	
48	3:57:31.873	canl	netfltl	8	212 06 00 00 00 00 00 00 00	
49	3:57:31.874	canl	TxCanl	8	112 07 00 00 00 00 00 00 00	
50	3:57:31.875	canl	netfltl	8	212 07 00 00 00 00 00 00 00	
51	3:57:31.876	canl	TxCanl	8	112 08 00 00 00 00 00 00 00	
52	3:57:31.876	canl	netfltl	8	212 08 00 00 00 00 00 00 00	
53	3:57:31.877	canl	TxCanl	8	112 09 00 00 00 00 00 00 00	
54	3:57:31.878	canl	netfltl	8	212 09 00 00 00 00 00 00 00	
55	3:57:31.879	canl	TxCanl	8	112 0A 00 00 00 00 00 00 00	
56	3:57:31.879	canl	netfltl	8	212 0A 00 00 00 00 00 00 00	
57	3:57:31.880	canl	TxCanl	8	112 0B 00 00 00 00 00 00 00	
58	3:57:31.881	canl	netfltl	8	212 0B 00 00 00 00 00 00 00	
59	3:57:31.882	canl	TxCanl	8	112 0C 00 00 00 00 00 00 00	
60	3:57:31.883	canl	netfltl	8	212 0C 00 00 00 00 00 00 00	
61	3:57:31.883	canl	TxCanl	8	112 0D 00 00 00 00 00 00 00	
62	3:57:31.884	canl	netfltl	8	212 0D 00 00 00 00 00 00 00	
63	3:57:31.885	canl	TxCanl	8	112 0E 00 00 00 00 00 00 00	
64	3:57:31.886	canl	netfltl	8	212 0E 00 00 00 00 00 00 00	
65	3:57:31.887	canl	TxCanl	8	112 0F 00 00 00 00 00 00 00	
66	3:57:31.887	canl	netfltl	8	212 0F 00 00 00 00 00 00 00	
67	3:57:31.888	ind			Event16 RTC: 14:01:25: D...	Start Step3 CAN...

Step 3: CAN1 CanFlt3 Test

Box1 user message3: can1->113 00 00 00 00 00 00 00 00
Box1 can2filter3: can1->213 -- -- -- -- -- -- -- --
Box1 sends can1->113 NN 00 00 00 00 00 00 00
Box2 responds with can1->213 NN 00 00 00 00 00 00 00 (can1flt3) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event16: Start Step 3

```
LOGEVNT; // log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //initialize x9 with count
SET umsgbyte30 = x9; //set user message3 byte 0 to x9
QUE usermsg3; //TX user message 3
SET timer1 = 100; //allow 100 ms for test
```

Event17: can1flt1 ->

```
SET x1 = rxdata0; //first data byte from RX message to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set y3 with flag corresponding to RX byte IF x1 == x9; //if RX byte equal TX byte
IF x9 < Fh; //if less than 16 messages has been received SET x9 + 1h; //increment TX byte SET umsgbyte30 = x9; //set first byte of user message3 to x9
    QUE usermsg3; //TX user message3
ELSE; //end of test
    IF y3 == FFFFh; //all messages have been received?
    CL offset+1; //go to step 4
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 3 Log:

Item	Time(h:m:s.ms)	Type	Filter	d..	Frame	Description
67	3:57:31.888	ind			Event16 RTC: 14:01:25: D...	Start Step3 CAN..
68	3:57:31.888	canl	TxCanl	8	113 00 00 00 00 00 00 00	
69	3:57:31.889	canl	canlflt03	8	213 00 00 00 00 00 00 00	
70	3:57:31.890	canl	TxCanl	8	113 01 00 00 00 00 00 00	
71	3:57:31.891	canl	canlflt03	8	213 01 00 00 00 00 00 00	
72	3:57:31.891	canl	TxCanl	8	113 02 00 00 00 00 00 00	
73	3:57:31.892	canl	canlflt03	8	213 02 00 00 00 00 00 00	
74	3:57:31.893	canl	TxCanl	8	113 03 00 00 00 00 00 00	
75	3:57:31.893	canl	canlflt03	8	213 03 00 00 00 00 00 00	
76	3:57:31.895	canl	TxCanl	8	113 04 00 00 00 00 00 00	
77	3:57:31.895	canl	canlflt03	8	213 04 00 00 00 00 00 00	
78	3:57:31.896	canl	TxCanl	8	113 05 00 00 00 00 00 00	
79	3:57:31.897	canl	canlflt03	8	213 05 00 00 00 00 00 00	
80	3:57:31.897	canl	TxCanl	8	113 06 00 00 00 00 00 00	
81	3:57:31.898	canl	canlflt03	8	213 06 00 00 00 00 00 00	
82	3:57:31.899	canl	TxCanl	8	113 07 00 00 00 00 00 00	
83	3:57:31.899	canl	canlflt03	8	213 07 00 00 00 00 00 00	
84	3:57:31.900	canl	TxCanl	8	113 08 00 00 00 00 00 00	
85	3:57:31.901	canl	canlflt03	8	213 08 00 00 00 00 00 00	
86	3:57:31.902	canl	TxCanl	8	113 09 00 00 00 00 00 00	
87	3:57:31.903	canl	canlflt03	8	213 09 00 00 00 00 00 00	
88	3:57:31.903	canl	TxCanl	8	113 0A 00 00 00 00 00 00	
89	3:57:31.904	canl	canlflt03	8	213 0A 00 00 00 00 00 00	
90	3:57:31.905	canl	TxCanl	8	113 0B 00 00 00 00 00 00	
91	3:57:31.906	canl	canlflt03	8	213 0B 00 00 00 00 00 00	
92	3:57:31.906	canl	TxCanl	8	113 0C 00 00 00 00 00 00	
93	3:57:31.907	canl	canlflt03	8	213 0C 00 00 00 00 00 00	
94	3:57:31.908	canl	TxCanl	8	113 0D 00 00 00 00 00 00	
95	3:57:31.909	canl	canlflt03	8	213 0D 00 00 00 00 00 00	
96	3:57:31.910	canl	TxCanl	8	113 0E 00 00 00 00 00 00	
97	3:57:31.910	canl	canlflt03	8	213 0E 00 00 00 00 00 00	
98	3:57:31.911	canl	TxCanl	8	113 0F 00 00 00 00 00 00	
99	3:57:31.912	canl	canlflt03	8	213 0F 00 00 00 00 00 00	

Step 4: Exd CAN1 UserFlt2 Test

Box1 user message4: can1->0000111 00 00 00 00 00 00 00 00
Box1 user filter2: can1->0000211 -- -- -- -- -- -- -- --
Box1 sends can1->0000111 NN 00 00 00 00 00 00 00
Box2 responds with can1->0000211 NN 00 00 00 00 00 00 00 (user filter2)
NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event18: Start Step 4

```
LOGEVNT; // log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //initialize x9 with count
SET umsgbyte40 = x9; //set user message4 byte 0 to x9
QUE usermsg4; //TX user message4
SET timer1 = 200; //allow 200 ms for test
```

Event19: filter2 ->

```
SET x1 = rxdata0; //first data byte from RX message to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set y3 with flag
corresponding to RX byte IF x1 == x9; //if RX byte equal TX byte
IF x9 < Fh; //if less than 16 messages has been received SET x9 + 1h;
//increment TX byte SET umsgbyte40 = x9; //set first byte of user message4
to x9
    QUE usermsg4; //TX user message4
ELSE; //end of test
    IF y3 == FFFFh; //all messages have been received?
    CL offset+1; //go to step 5
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 4 Log:

Item	Time(h:m:s.ms)	Type	Filter	d..	Frame	Description
100	3:57:31.912	ind			Event18 RTC: 14:01:25: D...	Start Step4 Exd..
101	3:57:31.913	canl	TxCanl	8	00000111 00 00 00 00 00 0...	
102	3:57:31.914	canl	flt02	8	00000211 00 00 00 00 00 0...	
103	3:57:31.914	canl	TxCanl	8	00000111 01 00 00 00 00 0...	
104	3:57:31.915	canl	flt02	8	00000211 01 00 00 00 00 0...	
105	3:57:31.916	canl	TxCanl	8	00000111 02 00 00 00 00 0...	
106	3:57:31.917	canl	flt02	8	00000211 02 00 00 00 00 0...	
107	3:57:31.918	canl	TxCanl	8	00000111 03 00 00 00 00 0...	
108	3:57:31.919	canl	flt02	8	00000211 03 00 00 00 00 0...	
109	3:57:31.920	canl	TxCanl	8	00000111 04 00 00 00 00 0...	
110	3:57:31.921	canl	flt02	8	00000211 04 00 00 00 00 0...	
111	3:57:31.922	canl	TxCanl	8	00000111 05 00 00 00 00 0...	
112	3:57:31.922	canl	flt02	8	00000211 05 00 00 00 00 0...	
113	3:57:31.923	canl	TxCanl	8	00000111 06 00 00 00 00 0...	
114	3:57:31.924	canl	flt02	8	00000211 06 00 00 00 00 0...	
115	3:57:31.925	canl	TxCanl	8	00000111 07 00 00 00 00 0...	
116	3:57:31.926	canl	flt02	8	00000211 07 00 00 00 00 0...	
117	3:57:31.927	canl	TxCanl	8	00000111 08 00 00 00 00 0...	
118	3:57:31.927	canl	flt02	8	00000211 08 00 00 00 00 0...	
119	3:57:31.928	canl	TxCanl	8	00000111 09 00 00 00 00 0...	
120	3:57:31.929	canl	flt02	8	00000211 09 00 00 00 00 0...	
121	3:57:31.930	canl	TxCanl	8	00000111 0A 00 00 00 00 0...	
122	3:57:31.931	canl	flt02	8	00000211 0A 00 00 00 00 0...	
123	3:57:31.932	canl	TxCanl	8	00000111 0B 00 00 00 00 0...	
124	3:57:31.933	canl	flt02	8	00000211 0B 00 00 00 00 0...	
125	3:57:31.933	canl	TxCanl	8	00000111 0C 00 00 00 00 0...	
126	3:57:31.934	canl	flt02	8	00000211 0C 00 00 00 00 0...	
127	3:57:31.935	canl	TxCanl	8	00000111 0D 00 00 00 00 0...	
128	3:57:31.936	canl	flt02	8	00000211 0D 00 00 00 00 0...	
129	3:57:31.937	canl	TxCanl	8	00000111 0E 00 00 00 00 0...	
130	3:57:31.938	canl	flt02	8	00000211 0E 00 00 00 00 0...	
131	3:57:31.939	canl	TxCanl	8	00000111 0F 00 00 00 00 0...	
132	3:57:31.939	canl	flt02	8	00000211 0F 00 00 00 00 0...	
133	3:57:31.940	ind			Event20 RTC: 14:01:25: D...	Start Step5 Ext..

Step 5: Exd CAN1 netflt2 Test

Box1 user message5: can1->0000112 00 00 00 00 00 00 00 00
Box1 network filter2: can1->0000212 -- -- -- -- -- -- -- --
Box1 sends can1->0000112 NN 00 00 00 00 00 00 00
Box2 responds with can1->0000212 NN 00 00 00 00 00 00 00 (network filter2)
NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event20: Start Step 5

```
LOGEVNT; // log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //initialize x9 with count
SET umsgbyte50 = x9; //set user message5 byte 0 to x9
QUE usermsg5; //TX user message5
SET timer1 = 200; //allow 200 ms for test
```

Event21: netflt2 ->

```
SET x1 = rxdata0; //first data byte from RX message to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set y3 with flag corresponding to RX byte IF x1 == x9; //if RX byte equal TX byte
IF x9 < Fh; //if less than 16 messages has been received SET x9 + 1h; //increment TX byte SET umsgbyte50 = x9; //set first byte of user message5 to x9
    QUE usermsg5; //TX user message5
ELSE; //end of test
    IF y3 == FFFFh; //all messages have been received?
    CL offset+1; //go to step 6
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 5 Log:

Item	Time(h:m:s.ms)	Type	Filter	d..	Frame	Description
133	3:57:31.940	ind			Event20 RTC: 14:01:25: D...	Start Step5 Ext..
134	3:57:31.940	canl	TxCanl	8	00000112 00 00 00 00 00 0...	
135	3:57:31.941	canl	netflt2	8	00000212 00 00 00 00 00 0...	
136	3:57:31.942	canl	TxCanl	8	00000112 01 00 00 00 00 0...	
137	3:57:31.943	canl	netflt2	8	00000212 01 00 00 00 00 0...	
138	3:57:31.944	canl	TxCanl	8	00000112 02 00 00 00 00 0...	
139	3:57:31.944	canl	netflt2	8	00000212 02 00 00 00 00 0...	
140	3:57:31.945	canl	TxCanl	8	00000112 03 00 00 00 00 0...	
141	3:57:31.946	canl	netflt2	8	00000212 03 00 00 00 00 0...	
142	3:57:31.946	canl	TxCanl	8	00000112 04 00 00 00 00 0...	
143	3:57:31.947	canl	netflt2	8	00000212 04 00 00 00 00 0...	
144	3:57:31.948	canl	TxCanl	8	00000112 05 00 00 00 00 0...	
145	3:57:31.949	canl	netflt2	8	00000212 05 00 00 00 00 0...	
146	3:57:31.950	canl	TxCanl	8	00000112 06 00 00 00 00 0...	
147	3:57:31.950	canl	netflt2	8	00000212 06 00 00 00 00 0...	
148	3:57:31.951	canl	TxCanl	8	00000112 07 00 00 00 00 0...	
149	3:57:31.952	canl	netflt2	8	00000212 07 00 00 00 00 0...	
150	3:57:31.953	canl	TxCanl	8	00000112 08 00 00 00 00 0...	
151	3:57:31.953	canl	netflt2	8	00000212 08 00 00 00 00 0...	
152	3:57:31.954	canl	TxCanl	8	00000112 09 00 00 00 00 0...	
153	3:57:31.955	canl	netflt2	8	00000212 09 00 00 00 00 0...	
154	3:57:31.956	canl	TxCanl	8	00000112 0A 00 00 00 00 0...	
155	3:57:31.957	canl	netflt2	8	00000212 0A 00 00 00 00 0...	
156	3:57:31.958	canl	TxCanl	8	00000112 0B 00 00 00 00 0...	
157	3:57:31.959	canl	netflt2	8	00000212 0B 00 00 00 00 0...	
158	3:57:31.959	canl	TxCanl	8	00000112 0C 00 00 00 00 0...	
159	3:57:31.960	canl	netflt2	8	00000212 0C 00 00 00 00 0...	
160	3:57:31.961	canl	TxCanl	8	00000112 0D 00 00 00 00 0...	
161	3:57:31.962	canl	netflt2	8	00000212 0D 00 00 00 00 0...	
162	3:57:31.963	canl	TxCanl	8	00000112 0E 00 00 00 00 0...	
163	3:57:31.964	canl	netflt2	8	00000212 0E 00 00 00 00 0...	
164	3:57:31.965	canl	TxCanl	8	00000112 0F 00 00 00 00 0...	
165	3:57:31.966	canl	netflt2	8	00000212 0F 00 00 00 00 0...	
166	3:57:31.966	ind			Event22 RTC: 14:01:25: D...	Start Step6 Ext..

Step 6: Exd CAN1 can1flt6 Test

Box1 user message6: can1->0000113 00 00 00 00 00 00 00 00
Box1 can1flt6: can1->0000213 -- -- -- -- -- -- -- --
Box1 sends can1->0000113 NN 00 00 00 00 00 00 00
Box2 responds with can1->0000213 NN 00 00 00 00 00 00 00 (can1flt6) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event22: Start Step 6

```
LOGEVNT; // log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //initialize x9 with count
SET umsgbyte60 = x9; //set user message6 byte 0 to x9
QUE usermsg6; //TX user message6
SET timer1 = 100; //allow 100 ms for test
```

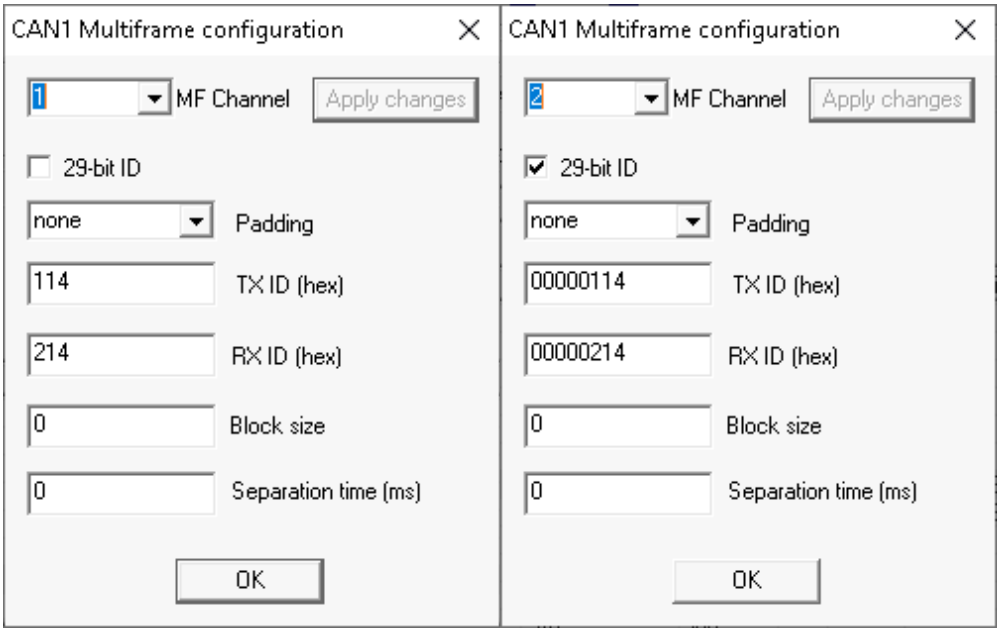
Event23: can1flt6 ->

```
SET x1 = rxdata0; //first data byte from RX message to x1;
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set y3 with flag corresponding to RX byte IF x1 == x9; //if RX byte equal TX byte
IF x9 < Fh; //if less than 16 messages has been received SET x9 + 1h; //increment TX byte SET umsgbyte60 = x9; //set first byte of user message6 to x9
    QUE usermsg6; //TX user message6
ELSE; //end of test
    IF y3 == FFFFh; //all messages have been received?
    CL offset+1; //go to step 7
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 6 Log:

Item	Time(h:m:s.ms)	Type	Filter	d..	Frame	Description
166	3:57:31.966	ind			Event22 RTC: 14:01:25: Date: ...	Start Step6
167	3:57:31.967	canl	TxCanl	8	00000113 00 00 00 00 00 00 00 00	
168	3:57:31.968	canl	canlflt06	8	00000213 00 00 00 00 00 00 00 C5	
169	3:57:31.968	canl	TxCanl	8	00000113 01 00 00 00 00 00 00 00	
170	3:57:31.969	canl	canlflt06	8	00000213 01 00 00 00 00 00 00 C5	
171	3:57:31.970	canl	TxCanl	8	00000113 02 00 00 00 00 00 00 00	
172	3:57:31.971	canl	canlflt06	8	00000213 02 00 00 00 00 00 00 C5	
173	3:57:31.971	canl	TxCanl	8	00000113 03 00 00 00 00 00 00 00	
174	3:57:31.972	canl	canlflt06	8	00000213 03 00 00 00 00 00 00 C5	
175	3:57:31.973	canl	TxCanl	8	00000113 04 00 00 00 00 00 00 00	
176	3:57:31.974	canl	canlflt06	8	00000213 04 00 00 00 00 00 00 C5	
177	3:57:31.975	canl	TxCanl	8	00000113 05 00 00 00 00 00 00 00	
178	3:57:31.976	canl	canlflt06	8	00000213 05 00 00 00 00 00 00 C5	
179	3:57:31.977	canl	TxCanl	8	00000113 06 00 00 00 00 00 00 00	
180	3:57:31.977	canl	canlflt06	8	00000213 06 00 00 00 00 00 00 C5	
181	3:57:31.978	canl	TxCanl	8	00000113 07 00 00 00 00 00 00 00	
182	3:57:31.979	canl	canlflt06	8	00000213 07 00 00 00 00 00 00 C5	
183	3:57:31.980	canl	TxCanl	8	00000113 08 00 00 00 00 00 00 00	
184	3:57:31.981	canl	canlflt06	8	00000213 08 00 00 00 00 00 00 C5	
185	3:57:31.982	canl	TxCanl	8	00000113 09 00 00 00 00 00 00 00	
186	3:57:31.982	canl	canlflt06	8	00000213 09 00 00 00 00 00 00 C5	
187	3:57:31.984	canl	TxCanl	8	00000113 0A 00 00 00 00 00 00 00	
188	3:57:31.984	canl	canlflt06	8	00000213 0A 00 00 00 00 00 00 C5	
189	3:57:31.985	canl	TxCanl	8	00000113 0B 00 00 00 00 00 00 00	
190	3:57:31.986	canl	canlflt06	8	00000213 0B 00 00 00 00 00 00 C5	
191	3:57:31.987	canl	TxCanl	8	00000113 0C 00 00 00 00 00 00 00	
192	3:57:31.987	canl	canlflt06	8	00000213 0C 00 00 00 00 00 00 C5	
193	3:57:31.988	canl	TxCanl	8	00000113 0D 00 00 00 00 00 00 00	
194	3:57:31.989	canl	canlflt06	8	00000213 0D 00 00 00 00 00 00 C5	
195	3:57:31.990	canl	TxCanl	8	00000113 0E 00 00 00 00 00 00 00	
196	3:57:31.990	canl	canlflt06	8	00000213 0E 00 00 00 00 00 00 C5	
197	3:57:31.991	canl	TxCanl	8	00000113 0F 00 00 00 00 00 00 00	
198	3:57:31.992	canl	canlflt06	8	00000213 0F 00 00 00 00 00 00 C5	
199	3:57:31.992	ind			Event24 RTC: 14:01:25: Date: ...	Start Step7

Step 7: Std CAN1 Multi-Frame Test



Multi-frame channel1 configuration for sub-channel 1,2 shown above.
Box1: fill out MF1 64-data bytes from umsgs 8,9,10: can1->114 10 11...4B 4C 4D 4E 4F
and TX MF1 to Box2.
Box2 detects MF1 and send back 64-byte message back: can1->214 -- -- -- -- -- -

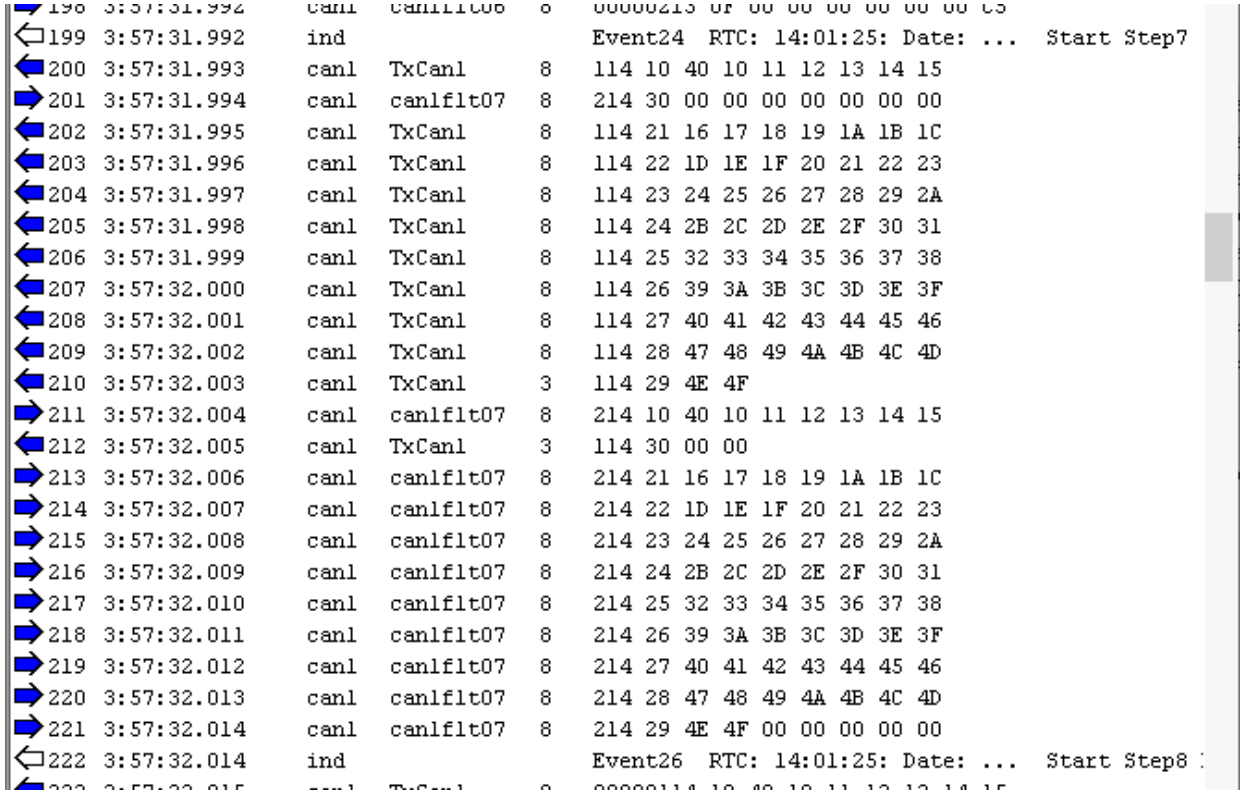
Event24: Start Step 7

```
LOGEVNT; // log event
SET umsgbyte80 = 10h; //restore original first user message8 byte SET timer1 = 100; //allow 100 ms for test
SET TxLong1 = bytesize64; //64-bytes for TX long message SET TxLong1 = usermsg8; //copy data bytes from user messages 8,9,10
SET mfCan101 = TxLong1; //assign TX long to MF1 CAN1 sub-channel 1
SET mfCan101 = bytesize64; //Set MF1 CAN1 to 64 bytes QUE mfCan101; //TX MF1 CAN1 sub-channel 1
```

Event25: mfCanRx101 ->

```
IF mfCan1 == usermsg8; //if received data equals to content of
usermsg8,9,10
CL offset+1; //go to step8
ELSE; //not equals
CL cl-tag4; //test failed
ENDIF;
```

Test Step 7 Log:



199	3:57:31.992	ind			Event24 RTC: 14:01:25: Date: ... Start Step7
200	3:57:31.993	canl	TxCanl	8	114 10 40 10 11 12 13 14 15
201	3:57:31.994	canl	canlflt07	8	214 30 00 00 00 00 00 00 00
202	3:57:31.995	canl	TxCanl	8	114 21 16 17 18 19 1A 1B 1C
203	3:57:31.996	canl	TxCanl	8	114 22 1D 1E 1F 20 21 22 23
204	3:57:31.997	canl	TxCanl	8	114 23 24 25 26 27 28 29 2A
205	3:57:31.998	canl	TxCanl	8	114 24 2B 2C 2D 2E 2F 30 31
206	3:57:31.999	canl	TxCanl	8	114 25 32 33 34 35 36 37 38
207	3:57:32.000	canl	TxCanl	8	114 26 39 3A 3B 3C 3D 3E 3F
208	3:57:32.001	canl	TxCanl	8	114 27 40 41 42 43 44 45 46
209	3:57:32.002	canl	TxCanl	8	114 28 47 48 49 4A 4B 4C 4D
210	3:57:32.003	canl	TxCanl	3	114 29 4E 4F
211	3:57:32.004	canl	canlflt07	8	214 10 40 10 11 12 13 14 15
212	3:57:32.005	canl	TxCanl	3	114 30 00 00
213	3:57:32.006	canl	canlflt07	8	214 21 16 17 18 19 1A 1B 1C
214	3:57:32.007	canl	canlflt07	8	214 22 1D 1E 1F 20 21 22 23
215	3:57:32.008	canl	canlflt07	8	214 23 24 25 26 27 28 29 2A
216	3:57:32.009	canl	canlflt07	8	214 24 2B 2C 2D 2E 2F 30 31
217	3:57:32.010	canl	canlflt07	8	214 25 32 33 34 35 36 37 38
218	3:57:32.011	canl	canlflt07	8	214 26 39 3A 3B 3C 3D 3E 3F
219	3:57:32.012	canl	canlflt07	8	214 27 40 41 42 43 44 45 46
220	3:57:32.013	canl	canlflt07	8	214 28 47 48 49 4A 4B 4C 4D
221	3:57:32.014	canl	canlflt07	8	214 29 4E 4F 00 00 00 00 00
222	3:57:32.014	ind			Event26 RTC: 14:01:25: Date: ... Start Step8

Step 8: Xtd CAN1 multi-frame test

Multi-frame channel1 configuration for sub-channel2 shown above.
Box1: fill out MF2 64-data bytes from umsgs 8,9,10: can1->0000114 10 11...4B 4C 4D 4E 4F
and TX MF2 to Box2.
Box2 detects MF2 and send back 64-byte message back: can1->0000214

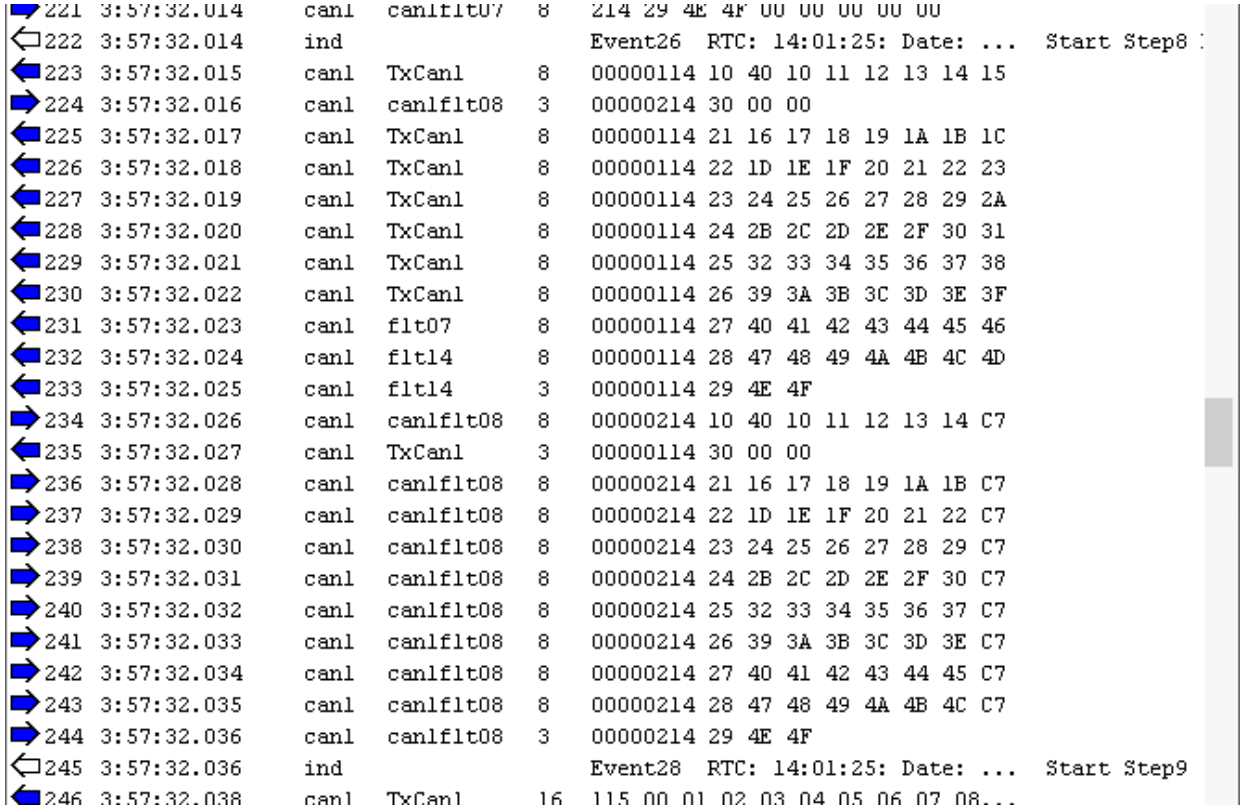
Event26: Start Step 8

```
LOGEVNT; // log event
SET timer1 = 100; //allow 100 ms for test
SET TxLong1 = bytesize64; //64-bytes for TX long message SET TxLong1 = usermsg8; //copy data bytes from user messages 8,9,10
SET mfCan201 = TxLong1; //assign TX long to MF2 CAN1 sub-channel 2
SET mfCan201 = bytesize64; //Set MF2 CAN1 to 64 bytes QUE mfCan201; //TX MF2 CAN1 sub-channel 2
```

Event27: mfCanRx201 ->

```
IF mfCan2 == usermsg8; //if received data equals to content of usermsg8,9,10
CL offset+1; //go to step8
ELSE; //not equals
CL cl-tag4; //test failed
ENDIF;
```

Test Step 8 log:



221	3:57:32.014	canl	canlflt07	8	214 29 4E 4F 00 00 00 00 00	
222	3:57:32.014	ind			Event26 RTC: 14:01:25: Date: ...	Start Step8
223	3:57:32.015	canl	TxCanl	8	00000114 10 40 10 11 12 13 14 15	
224	3:57:32.016	canl	canlflt08	3	00000214 30 00 00	
225	3:57:32.017	canl	TxCanl	8	00000114 21 16 17 18 19 1A 1B 1C	
226	3:57:32.018	canl	TxCanl	8	00000114 22 1D 1E 1F 20 21 22 23	
227	3:57:32.019	canl	TxCanl	8	00000114 23 24 25 26 27 28 29 2A	
228	3:57:32.020	canl	TxCanl	8	00000114 24 2B 2C 2D 2E 2F 30 31	
229	3:57:32.021	canl	TxCanl	8	00000114 25 32 33 34 35 36 37 38	
230	3:57:32.022	canl	TxCanl	8	00000114 26 39 3A 3B 3C 3D 3E 3F	
231	3:57:32.023	canl	flt07	8	00000114 27 40 41 42 43 44 45 46	
232	3:57:32.024	canl	flt14	8	00000114 28 47 48 49 4A 4B 4C 4D	
233	3:57:32.025	canl	flt14	3	00000114 29 4E 4F	
234	3:57:32.026	canl	canlflt08	8	00000214 10 40 10 11 12 13 14 C7	
235	3:57:32.027	canl	TxCanl	3	00000114 30 00 00	
236	3:57:32.028	canl	canlflt08	8	00000214 21 16 17 18 19 1A 1B C7	
237	3:57:32.029	canl	canlflt08	8	00000214 22 1D 1E 1F 20 21 22 C7	
238	3:57:32.030	canl	canlflt08	8	00000214 23 24 25 26 27 28 29 C7	
239	3:57:32.031	canl	canlflt08	8	00000214 24 2B 2C 2D 2E 2F 30 C7	
240	3:57:32.032	canl	canlflt08	8	00000214 25 32 33 34 35 36 37 C7	
241	3:57:32.033	canl	canlflt08	8	00000214 26 39 3A 3B 3C 3D 3E C7	
242	3:57:32.034	canl	canlflt08	8	00000214 27 40 41 42 43 44 45 C7	
243	3:57:32.035	canl	canlflt08	8	00000214 28 47 48 49 4A 4B 4C C7	
244	3:57:32.036	canl	canlflt08	3	00000214 29 4E 4F	
245	3:57:32.036	ind			Event28 RTC: 14:01:25: Date: ...	Start Step9
246	3:57:32.038	canl	TxCanl	16	115 00 01 02 03 04 05 06 07 08...	

Step 9: Std CAN1 FD Test

Box1: TX usermsg11 (CAN FD) can1->115 00 01 ...3E 3F (64 bytes) Box2: Responds with (CAN FD) can1->215 00 01...3E 3F (64 bytes) can1flt14
Box1 verifies selected RX bytes and if match test pass

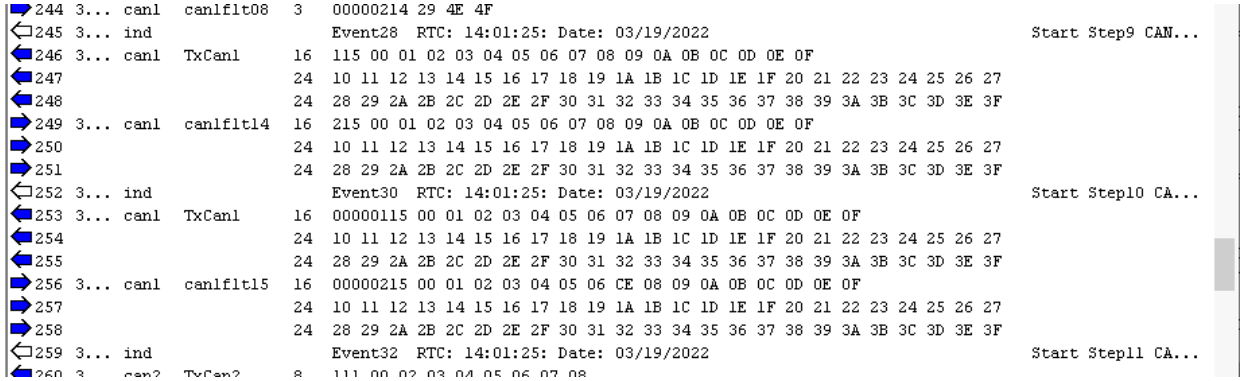
Event28: Start Step 9

```
SET timer1 = 100; //allow 100 ms for test
IF deviceState18 > 1Dh; //if hardware is NW92 and above LOGEVNT; //log event
QUE usermsg11; //TX CAN FD message
ELSE; //if not CAN FD hardware
CL cl-tag32; //skip CAN FD test, go to step 11
ENDIF;
```

Event29: can1flt14 ->

```
IF rxdata15 != Fh; //verify RX byte 16
SET x1 = Fh; SET x2 = rxdata15; CL cl-tag5; //byte 16 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata39 != 27h; //verify RX byte 40
SET x1 = 27h; SET x2 = rxdata39; CL cl-tag5; //byte 40 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata63 != 3Fh; //verify RX byte 64
SET x1 = 3Fh; SET x2 = rxdata63; CL cl-tag5; //byte 64 not valid, test failed
ENDCL; //stop verification
ENDIF;
CL offset+1; //Ok, go to step 10
```

Test Step 9, 10 log:



```
244 3... canl   canlflt08   3   00000214 29 4E 4F
245 3... ind                    Event28  RTC: 14:01:25: Date: 03/19/2022                                  Start Step9 CAN...
246 3... canl   TxCanl      16  115 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
247                         24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
248                         24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
249 3... canl   canlfltl4   16  215 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
250                         24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
251                         24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
252 3... ind                    Event30  RTC: 14:01:25: Date: 03/19/2022                                  Start Step10 CA...
253 3... canl   TxCanl      16  00000115 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
254                         24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
255                         24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
256 3... canl   canlfltl5   16  00000215 00 01 02 03 04 05 06 CE 08 09 0A 0B 0C 0D 0E 0F
257                         24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
258                         24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
259 3... ind                    Event32  RTC: 14:01:25: Date: 03/19/2022                                  Start Step11 CA...
260 3... can2   TxCan2      8   111 00 02 03 04 05 06 07 08
```

Step 10: Xtd CAN1 FD Test

Box1: TX usermsg15 (CAN FD) can1->0000115 00 01 ...3E 3F (64 bytes)
Box2: Responds with (CAN FD) can1->0000215 00 01...3E 3F (64 bytes) can1flt15
Box1 verifies selected RX bytes and if match test pass

Event30: Start Step 10

```
SET timer1 = 100; //allow 100 ms for test
IF deviceState18 > 1Dh; //if hardware is NW92 and above LOGEVNT; //
log event
QUE usermsg15; //TX XTD CAN FD message
ELSE; //if not CAN FD hardware
CL cl-tag32; //skip CAN FD test, go to step 11
ENDIF;
```

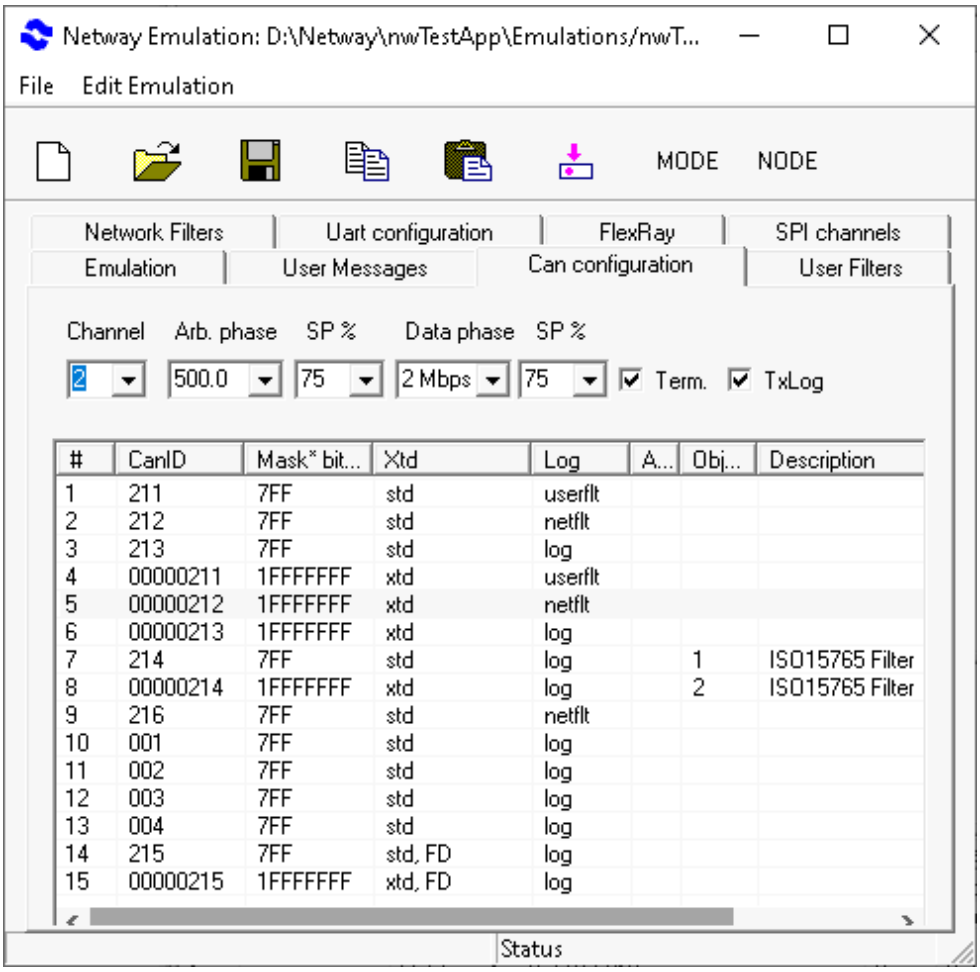
Event31: can1flt15 ->

```
IF rxdata15 != Fh; //verify RX byte 16
SET x1 = Fh; SET x2 = rxdata15; CL cl-tag5; //byte 16 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata39 != 27h; //verify RX byte 40
SET x1 = 27h; SET x2 = rxdata39; CL cl-tag5; //byte 40 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata63 != 3Fh; //verify RX byte 64
SET x1 = 3Fh; SET x2 = rxdata63; CL cl-tag5; //byte 64 not valid, test failed
ENDCL; //stop verification
ENDIF;
CL offset+1; //Ok, go to step 11
```

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights

reserved.

CAN2 Test Steps



#	CanID	Mask* bit...	Xtd	Log	A...	Obj...	Description
1	211	7FF	std	userflt			
2	212	7FF	std	netflt			
3	213	7FF	std	log			
4	00000211	1FFFFFFF	xtd	userflt			
5	00000212	1FFFFFFF	xtd	netflt			
6	00000213	1FFFFFFF	xtd	log			
7	214	7FF	std	log		1	ISO15765 Filter
8	00000214	1FFFFFFF	xtd	log		2	ISO15765 Filter
9	216	7FF	std	netflt			
10	001	7FF	std	log			
11	002	7FF	std	log			
12	003	7FF	std	log			
13	004	7FF	std	log			
14	215	7FF	std, FD	log			
15	00000215	1FFFFFFF	xtd, FD	log			

CAN configuration for channel 2 is shown above.

Step 11: CAN2 Filter3 Test

Box1 user message21: can2->111 00 00 00 00 00 00 00 00
Box1 filter3: can2->211 -- -- -- -- -- -- -- --
Box1 sends can2->111 NN 00 00 00 00 00 00 00
Box2 responds with can2->211 NN 00 00 00 00 00 00 00 (filter3) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event32: Start Step 11

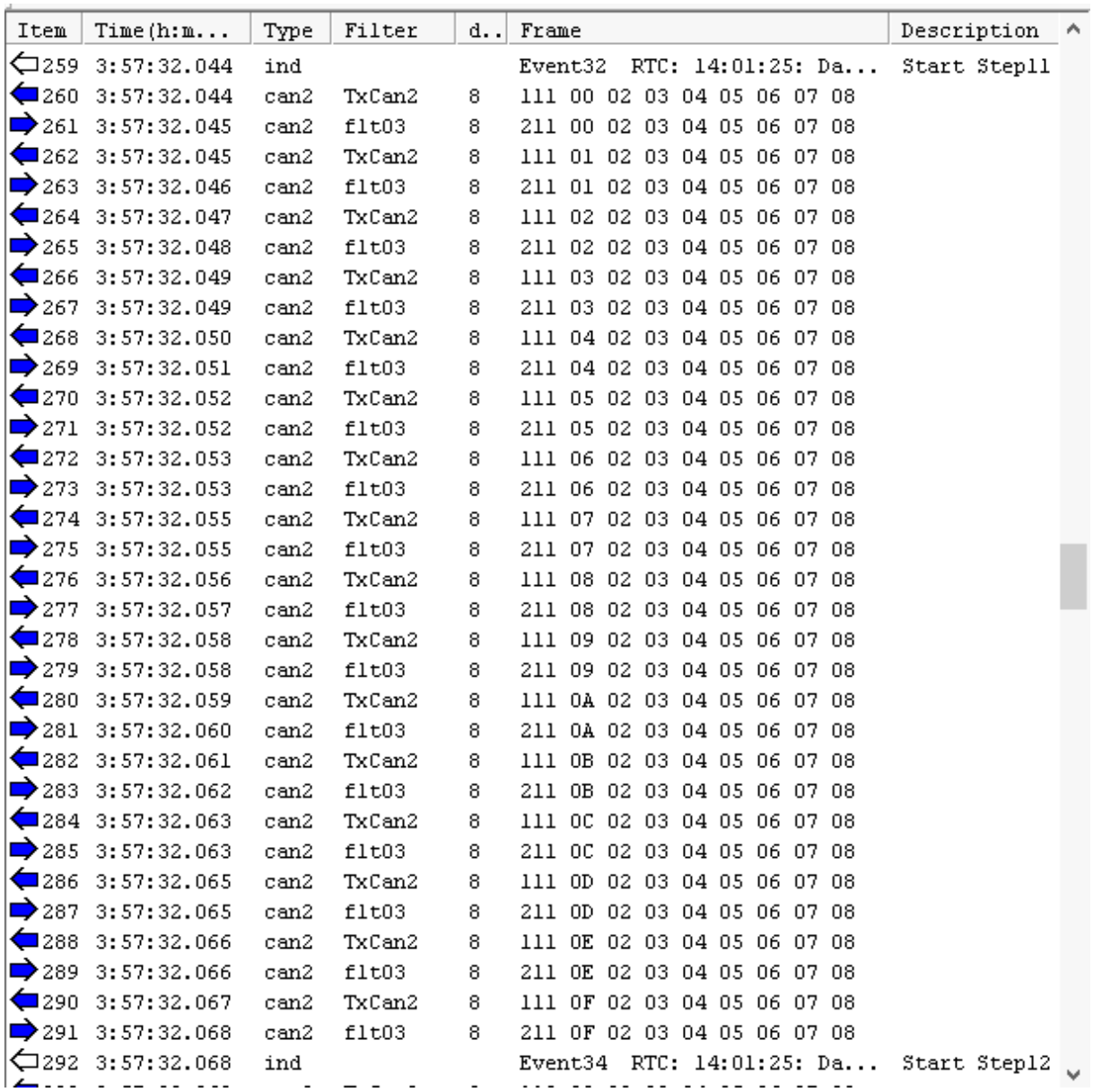
```
IF protocol6 == 0h; //if CAN2 is not available
CL cl-tag10; //test failed
ENDCL; //stop test
ENDIF;
LOGEVNT; //log event
SET x1 = 0h; //initialize count
SET x9 = 0h; //initialize x9
SET umsgbyte210 = x9; //x9 to first byte of user message21
QUE usermsg21; //TX user message21
SET y3 = 0h; //initialize y3 - RX messages flags
SET timer1 = 100; //allow 100 ms for this test
```

Event33: filter3 ->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask flag for RX message per x1
IF x1 == x9; //if RX byte equals TX byte
IF x9 < Fh; //if less than 15 messages received
    SET x9 + 1h; //increment RX count SET umsgbyte210 = x9; //put it into
 first byte of user message21
    QUE usermsg21; //TX user message21
ELSE; //end of test
    IF y3 == FFFFh; //all messages received?
    CL offset+1; //go to test step 12
    ELSE; //not all messages received
```

CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;

Test Step 11 Log:



Item	Time(h:m...	Type	Filter	d..	Frame	Description
259	3:57:32.044	ind			Event32 RTC: 14:01:25: Da...	Start Step11
260	3:57:32.044	can2	TxCan2	8	111 00 02 03 04 05 06 07 08	
261	3:57:32.045	can2	flt03	8	211 00 02 03 04 05 06 07 08	
262	3:57:32.045	can2	TxCan2	8	111 01 02 03 04 05 06 07 08	
263	3:57:32.046	can2	flt03	8	211 01 02 03 04 05 06 07 08	
264	3:57:32.047	can2	TxCan2	8	111 02 02 03 04 05 06 07 08	
265	3:57:32.048	can2	flt03	8	211 02 02 03 04 05 06 07 08	
266	3:57:32.049	can2	TxCan2	8	111 03 02 03 04 05 06 07 08	
267	3:57:32.049	can2	flt03	8	211 03 02 03 04 05 06 07 08	
268	3:57:32.050	can2	TxCan2	8	111 04 02 03 04 05 06 07 08	
269	3:57:32.051	can2	flt03	8	211 04 02 03 04 05 06 07 08	
270	3:57:32.052	can2	TxCan2	8	111 05 02 03 04 05 06 07 08	
271	3:57:32.052	can2	flt03	8	211 05 02 03 04 05 06 07 08	
272	3:57:32.053	can2	TxCan2	8	111 06 02 03 04 05 06 07 08	
273	3:57:32.053	can2	flt03	8	211 06 02 03 04 05 06 07 08	
274	3:57:32.055	can2	TxCan2	8	111 07 02 03 04 05 06 07 08	
275	3:57:32.055	can2	flt03	8	211 07 02 03 04 05 06 07 08	
276	3:57:32.056	can2	TxCan2	8	111 08 02 03 04 05 06 07 08	
277	3:57:32.057	can2	flt03	8	211 08 02 03 04 05 06 07 08	
278	3:57:32.058	can2	TxCan2	8	111 09 02 03 04 05 06 07 08	
279	3:57:32.058	can2	flt03	8	211 09 02 03 04 05 06 07 08	
280	3:57:32.059	can2	TxCan2	8	111 0A 02 03 04 05 06 07 08	
281	3:57:32.060	can2	flt03	8	211 0A 02 03 04 05 06 07 08	
282	3:57:32.061	can2	TxCan2	8	111 0B 02 03 04 05 06 07 08	
283	3:57:32.062	can2	flt03	8	211 0B 02 03 04 05 06 07 08	
284	3:57:32.063	can2	TxCan2	8	111 0C 02 03 04 05 06 07 08	
285	3:57:32.063	can2	flt03	8	211 0C 02 03 04 05 06 07 08	
286	3:57:32.065	can2	TxCan2	8	111 0D 02 03 04 05 06 07 08	
287	3:57:32.065	can2	flt03	8	211 0D 02 03 04 05 06 07 08	
288	3:57:32.066	can2	TxCan2	8	111 0E 02 03 04 05 06 07 08	
289	3:57:32.066	can2	flt03	8	211 0E 02 03 04 05 06 07 08	
290	3:57:32.067	can2	TxCan2	8	111 0F 02 03 04 05 06 07 08	
291	3:57:32.068	can2	flt03	8	211 0F 02 03 04 05 06 07 08	
292	3:57:32.068	ind			Event34 RTC: 14:01:25: Da...	Start Step12

Step 12: CAN2 NetFlt3 Test

Box1 user message22: can2->112 00 00 00 00 00 00 00 00
Box1 filter3: can2->212 -- -- -- -- -- -- -- --
Box1 sends can2->112 NN 00 00 00 00 00 00 00
Box2 responds with can2->212 NN 00 00 00 00 00 00 00 (netflt3) NN = 0
-:- 0F. 16 messages transmitted, 16 messages received

Event34: Start Step 12

```
LOGEVNT; //log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //RX message mask to 0
SET umsgbyte220 = x9; //first byte of user message22 = x9
QUE usermsg22; //TX user message22
SET timer1 = 100; //allow 100 ms for test
```

Event35: netflt3 ->

```
SET x1 = rxdata0; //RX first byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask flag for RX message
IF x1 == x9; //if RX byte equals TX byte
IF x9 < Fh; //if not all messages transmitted
    SET x9 + 1h; //increment message count SET umsgbyte220 = x9; //put it
 as first byte in user message22
    QUE usermsg22; //TX user message22
ELSE; //end of test
    IF y3 == FFFFh; //all messages received CL offset+1; //go to next test
 step13
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 12 Log:

Item	Time(h:m...	Type	Filter	d..	Frame	Description
292	3:57:32.068	ind			Event34 RTC: 14:01:25: Da...	Start Step12
293	3:57:32.069	can2	TxCan2	8	112 00 02 03 04 05 06 07 08	
294	3:57:32.070	can2	netflt3	8	212 00 02 03 04 05 06 07 08	
295	3:57:32.070	can2	TxCan2	8	112 01 02 03 04 05 06 07 08	
296	3:57:32.071	can2	netflt3	8	212 01 02 03 04 05 06 07 08	
297	3:57:32.072	can2	TxCan2	8	112 02 02 03 04 05 06 07 08	
298	3:57:32.072	can2	netflt3	8	212 02 02 03 04 05 06 07 08	
299	3:57:32.073	can2	TxCan2	8	112 03 02 03 04 05 06 07 08	
300	3:57:32.074	can2	netflt3	8	212 03 02 03 04 05 06 07 08	
301	3:57:32.074	can2	TxCan2	8	112 04 02 03 04 05 06 07 08	
302	3:57:32.075	can2	netflt3	8	212 04 02 03 04 05 06 07 08	
303	3:57:32.076	can2	TxCan2	8	112 05 02 03 04 05 06 07 08	
304	3:57:32.077	can2	netflt3	8	212 05 02 03 04 05 06 07 08	
305	3:57:32.078	can2	TxCan2	8	112 06 02 03 04 05 06 07 08	
306	3:57:32.078	can2	netflt3	8	212 06 02 03 04 05 06 07 08	
307	3:57:32.079	can2	TxCan2	8	112 07 02 03 04 05 06 07 08	
308	3:57:32.080	can2	netflt3	8	212 07 02 03 04 05 06 07 08	
309	3:57:32.081	can2	TxCan2	8	112 08 02 03 04 05 06 07 08	
310	3:57:32.081	can2	netflt3	8	212 08 02 03 04 05 06 07 08	
311	3:57:32.082	can2	TxCan2	8	112 09 02 03 04 05 06 07 08	
312	3:57:32.083	can2	netflt3	8	212 09 02 03 04 05 06 07 08	
313	3:57:32.084	can2	TxCan2	8	112 0A 02 03 04 05 06 07 08	
314	3:57:32.084	can2	netflt3	8	212 0A 02 03 04 05 06 07 08	
315	3:57:32.085	can2	TxCan2	8	112 0B 02 03 04 05 06 07 08	
316	3:57:32.086	can2	netflt3	8	212 0B 02 03 04 05 06 07 08	
317	3:57:32.087	can2	TxCan2	8	112 0C 02 03 04 05 06 07 08	
318	3:57:32.087	can2	netflt3	8	212 0C 02 03 04 05 06 07 08	
319	3:57:32.088	can2	TxCan2	8	112 0D 02 03 04 05 06 07 08	
320	3:57:32.089	can2	netflt3	8	212 0D 02 03 04 05 06 07 08	
321	3:57:32.090	can2	TxCan2	8	112 0E 02 03 04 05 06 07 08	
322	3:57:32.091	can2	netflt3	8	212 0E 02 03 04 05 06 07 08	
323	3:57:32.091	can2	TxCan2	8	112 0F 02 03 04 05 06 07 08	
324	3:57:32.092	can2	netflt3	8	212 0F 02 03 04 05 06 07 08	
325	3:57:32.092	ind			Event36 RTC: 14:01:25: Da...	Start Step13

Step 13: CAN2 can2flt3 Test

Box1 user message23: can2->113 00 00 00 00 00 00 00 00
Box1 can2flt3: can2->213 -- -- -- -- -- -- -- --
Box1 sends can2->113 NN 00 00 00 00 00 00 00
Box2 responds with can2->213 NN 00 00 00 00 00 00 00 (can2flt3) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event36: Start Step 13

```
LOGEVNT; //log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //RX message mask to 0
SET umsgbyte230 = x9; //first byte of user message23 = x9
QUE usermsg23; //TX user message23
SET timer1 = 100; //allow 100 ms for test
```

Event37: can2flt3 ->

```
SET x1 = rxdata0; //RX first byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask flag for RX message
IF x1 == x9; //if RX byte equals TX byte
IF x9 < Fh; //if not all messages transmitted
    SET x9 + 1h; //increment message count SET umsgbyte230 = x9; //put it
 as first byte in user message23
    QUE usermsg23; //TX user message23
ELSE; //end of test
    IF y3 == FFFFh; //all messages received CL offset+1; //go to next test
 step14
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 13 Log:

Item	Time(h:m...	Type	Filter	d..	Frame	Description
325	3:57:32.092	ind			Event36 RTC: 14:01:25: Da...	Start Step13
326	3:57:32.093	can2	TxCan2	8	113 00 02 03 04 05 06 07 08	
327	3:57:32.093	can2	can2f...	8	213 00 02 03 04 05 06 07 08	
328	3:57:32.094	can2	TxCan2	8	113 01 02 03 04 05 06 07 08	
329	3:57:32.095	can2	can2f...	8	213 01 02 03 04 05 06 07 08	
330	3:57:32.096	can2	TxCan2	8	113 02 02 03 04 05 06 07 08	
331	3:57:32.097	can2	can2f...	8	213 02 02 03 04 05 06 07 08	
332	3:57:32.097	can2	TxCan2	8	113 03 02 03 04 05 06 07 08	
333	3:57:32.098	can2	can2f...	8	213 03 02 03 04 05 06 07 08	
334	3:57:32.099	can2	TxCan2	8	113 04 02 03 04 05 06 07 08	
335	3:57:32.100	can2	can2f...	8	213 04 02 03 04 05 06 07 08	
336	3:57:32.100	can2	TxCan2	8	113 05 02 03 04 05 06 07 08	
337	3:57:32.101	can2	can2f...	8	213 05 02 03 04 05 06 07 08	
338	3:57:32.102	can2	TxCan2	8	113 06 02 03 04 05 06 07 08	
339	3:57:32.102	can2	can2f...	8	213 06 02 03 04 05 06 07 08	
340	3:57:32.103	can2	TxCan2	8	113 07 02 03 04 05 06 07 08	
341	3:57:32.104	can2	can2f...	8	213 07 02 03 04 05 06 07 08	
342	3:57:32.105	can2	TxCan2	8	113 08 02 03 04 05 06 07 08	
343	3:57:32.105	can2	can2f...	8	213 08 02 03 04 05 06 07 08	
344	3:57:32.106	can2	TxCan2	8	113 09 02 03 04 05 06 07 08	
345	3:57:32.107	can2	can2f...	8	213 09 02 03 04 05 06 07 08	
346	3:57:32.108	can2	TxCan2	8	113 0A 02 03 04 05 06 07 08	
347	3:57:32.108	can2	can2f...	8	213 0A 02 03 04 05 06 07 08	
348	3:57:32.109	can2	TxCan2	8	113 0B 02 03 04 05 06 07 08	
349	3:57:32.110	can2	can2f...	8	213 0B 02 03 04 05 06 07 08	
350	3:57:32.111	can2	TxCan2	8	113 0C 02 03 04 05 06 07 08	
351	3:57:32.111	can2	can2f...	8	213 0C 02 03 04 05 06 07 08	
352	3:57:32.112	can2	TxCan2	8	113 0D 02 03 04 05 06 07 08	
353	3:57:32.113	can2	can2f...	8	213 0D 02 03 04 05 06 07 08	
354	3:57:32.114	can2	TxCan2	8	113 0E 02 03 04 05 06 07 08	
355	3:57:32.114	can2	can2f...	8	213 0E 02 03 04 05 06 07 08	
356	3:57:32.115	can2	TxCan2	8	113 0F 02 03 04 05 06 07 08	
357	3:57:32.116	can2	can2f...	8	213 0F 02 03 04 05 06 07 08	
358	3:57:32.116	ind			Event38 RTC: 14:01:25: Da...	Start Step14

Step 14: Xtd CAN2 User Filter4 Test

Box1 user message24: can1->0000111 00 02 03 04 05 06 07 08
Box1 user filter4: can1->0000211 -- -- -- -- -- -- -- --
Box1 sends can1->0000111 NN 00 00 00 00 00 00 00
Box2 responds with can1->0000211 NN 00 00 00 00 00 00 00 (user filter4)
NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event38: Start Step 14

```
LOGEVNT; //log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //RX messages mask = 0
SET umsgbyte240 = x9; //set first byte of user message24 to x9
QUE usermsg24; //TX user message24
SET timer1 = 100; //allow 100 ms for test
```

Event39: filter4 ->

```
SET x1 = rxdata0; //RX first byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask flag for RX message
IF x1 == x9; //if RX byte equals TX byte
IF x9 < Fh; //if not all messages transmitted
    SET x9 + 1h; //increment message count SET umsgbyte240 = x9; //put it
 as first byte in user message24
    QUE usermsg24; //TX user message24
ELSE; //end of test
    IF y3 == FFFFh; //all messages received CL offset+1; //go to next test
 step15
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 14 Log:

Item	Time(h:m...	Type	Filter	d..	Frame	Description
358	3:57:32.116	ind			Event38 RTC: 14:01:25: Date: ...	Start Step14
359	3:57:32.117	can2	TxCan2	8	00000111 00 02 03 04 05 06 07 08	
360	3:57:32.118	can2	flt04	8	00000211 00 02 03 04 05 06 07 43	
361	3:57:32.118	can2	TxCan2	8	00000111 01 02 03 04 05 06 07 08	
362	3:57:32.119	can2	flt04	8	00000211 01 02 03 04 05 06 07 43	
363	3:57:32.120	can2	TxCan2	8	00000111 02 02 03 04 05 06 07 08	
364	3:57:32.121	can2	flt04	8	00000211 02 02 03 04 05 06 07 43	
365	3:57:32.122	can2	TxCan2	8	00000111 03 02 03 04 05 06 07 08	
366	3:57:32.123	can2	flt04	8	00000211 03 02 03 04 05 06 07 43	
367	3:57:32.124	can2	TxCan2	8	00000111 04 02 03 04 05 06 07 08	
368	3:57:32.124	can2	flt04	8	00000211 04 02 03 04 05 06 07 43	
369	3:57:32.125	can2	TxCan2	8	00000111 05 02 03 04 05 06 07 08	
370	3:57:32.126	can2	flt04	8	00000211 05 02 03 04 05 06 07 43	
371	3:57:32.127	can2	TxCan2	8	00000111 06 02 03 04 05 06 07 08	
372	3:57:32.128	can2	flt04	8	00000211 06 02 03 04 05 06 07 43	
373	3:57:32.128	can2	TxCan2	8	00000111 07 02 03 04 05 06 07 08	
374	3:57:32.129	can2	flt04	8	00000211 07 02 03 04 05 06 07 43	
375	3:57:32.131	can2	TxCan2	8	00000111 08 02 03 04 05 06 07 08	
376	3:57:32.131	can2	flt04	8	00000211 08 02 03 04 05 06 07 43	
377	3:57:32.132	can2	TxCan2	8	00000111 09 02 03 04 05 06 07 08	
378	3:57:32.133	can2	flt04	8	00000211 09 02 03 04 05 06 07 43	
379	3:57:32.134	can2	TxCan2	8	00000111 0A 02 03 04 05 06 07 08	
380	3:57:32.135	can2	flt04	8	00000211 0A 02 03 04 05 06 07 43	
381	3:57:32.136	can2	TxCan2	8	00000111 0B 02 03 04 05 06 07 08	
382	3:57:32.137	can2	flt04	8	00000211 0B 02 03 04 05 06 07 43	
383	3:57:32.137	can2	TxCan2	8	00000111 0C 02 03 04 05 06 07 08	
384	3:57:32.138	can2	flt04	8	00000211 0C 02 03 04 05 06 07 43	
385	3:57:32.139	can2	TxCan2	8	00000111 0D 02 03 04 05 06 07 08	
386	3:57:32.140	can2	flt04	8	00000211 0D 02 03 04 05 06 07 43	
387	3:57:32.141	can2	TxCan2	8	00000111 0E 02 03 04 05 06 07 08	
388	3:57:32.142	can2	flt04	8	00000211 0E 02 03 04 05 06 07 43	
389	3:57:32.143	can2	TxCan2	8	00000111 0F 02 03 04 05 06 07 08	
390	3:57:32.144	can2	flt04	8	00000211 0F 02 03 04 05 06 07 43	
391	3:57:32.144	ind			Event40 RTC: 14:01:25: Date: ...	Start Step15

Step 15: Xtd CAN2 Network Filter4 Test

Box1 user message25: can1->0000112 00 02 03 04 05 06 07 08
Box1 network filter4: can1->0000212 -- -- -- -- -- -- -- --
Box1 sends can1->0000112 NN 00 00 00 00 00 00 00
Box2 responds with can1->0000213 NN 00 00 00 00 00 00 00 (network filter4)
NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

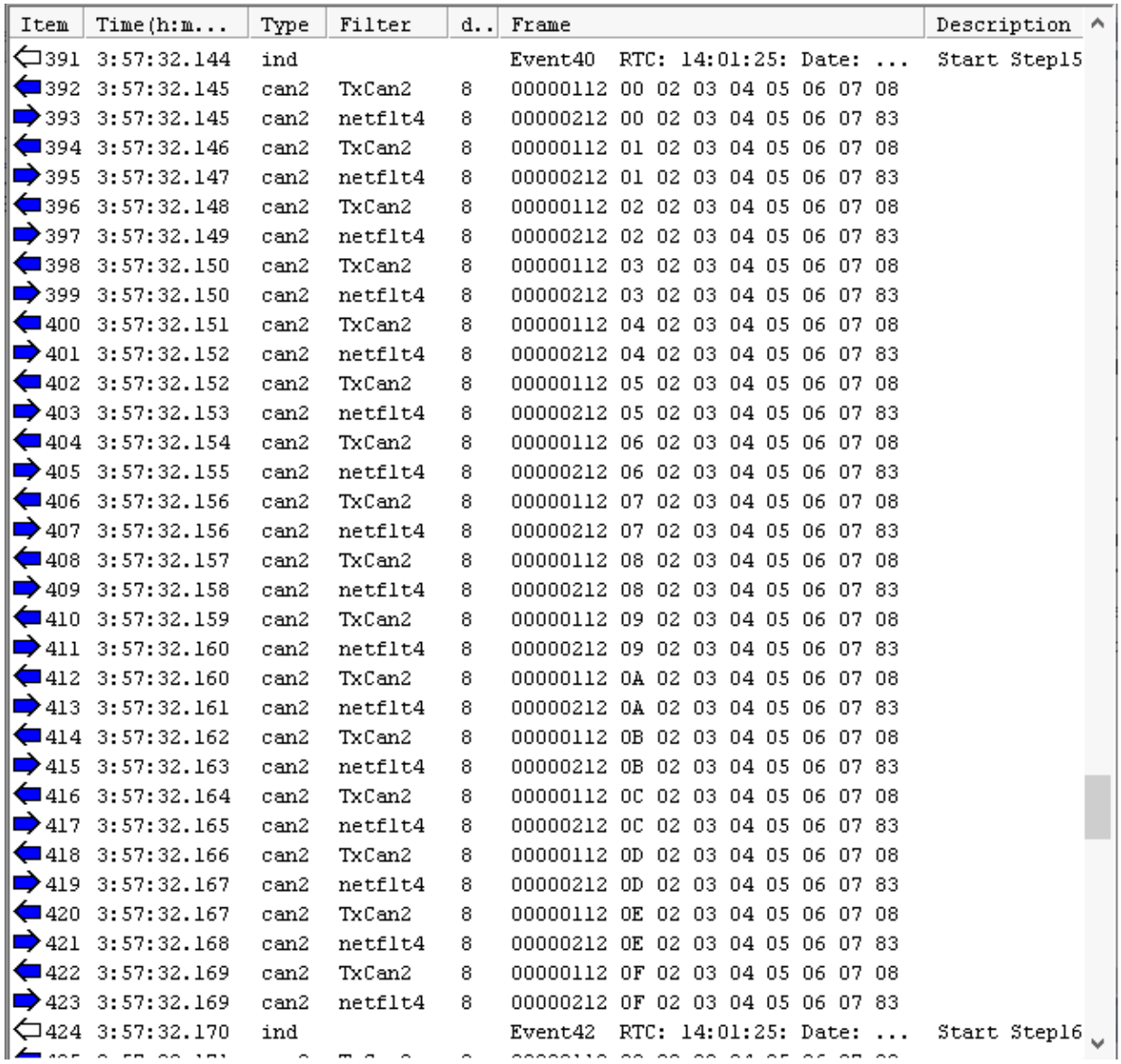
Event40: Start Step 15

```
LOGEVNT; //log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //RX messages mask = 0
SET umsgbyte250 = x9; //set first byte of user message25 to x9
QUE usermsg25; //TX user message25
SET timer1 = 100; //allow 100 ms for test
```

Event41: netflt4 ->

```
SET x1 = rxdata0; //RX first byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask flag for RX message
IF x1 == x9; //if RX byte equals TX byte
IF x9 < Fh; //if not all messages transmitted
    SET x9 + 1h; //increment message count SET umsgbyte250 = x9; //put it
 as first byte in user message25
    QUE usermsg25; //TX user message25
ELSE; //end of test
    IF y3 == FFFFh; //all messages received CL offset+1; //go to next test
 step16
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 15 Log:



Item	Time(h:m...	Type	Filter	d..	Frame	Description
391	3:57:32.144	ind			Event40 RTC: 14:01:25: Date: ...	Start Step15
392	3:57:32.145	can2	TxCan2	8	00000112 00 02 03 04 05 06 07 08	
393	3:57:32.145	can2	netflt4	8	00000212 00 02 03 04 05 06 07 83	
394	3:57:32.146	can2	TxCan2	8	00000112 01 02 03 04 05 06 07 08	
395	3:57:32.147	can2	netflt4	8	00000212 01 02 03 04 05 06 07 83	
396	3:57:32.148	can2	TxCan2	8	00000112 02 02 03 04 05 06 07 08	
397	3:57:32.149	can2	netflt4	8	00000212 02 02 03 04 05 06 07 83	
398	3:57:32.150	can2	TxCan2	8	00000112 03 02 03 04 05 06 07 08	
399	3:57:32.150	can2	netflt4	8	00000212 03 02 03 04 05 06 07 83	
400	3:57:32.151	can2	TxCan2	8	00000112 04 02 03 04 05 06 07 08	
401	3:57:32.152	can2	netflt4	8	00000212 04 02 03 04 05 06 07 83	
402	3:57:32.152	can2	TxCan2	8	00000112 05 02 03 04 05 06 07 08	
403	3:57:32.153	can2	netflt4	8	00000212 05 02 03 04 05 06 07 83	
404	3:57:32.154	can2	TxCan2	8	00000112 06 02 03 04 05 06 07 08	
405	3:57:32.155	can2	netflt4	8	00000212 06 02 03 04 05 06 07 83	
406	3:57:32.156	can2	TxCan2	8	00000112 07 02 03 04 05 06 07 08	
407	3:57:32.156	can2	netflt4	8	00000212 07 02 03 04 05 06 07 83	
408	3:57:32.157	can2	TxCan2	8	00000112 08 02 03 04 05 06 07 08	
409	3:57:32.158	can2	netflt4	8	00000212 08 02 03 04 05 06 07 83	
410	3:57:32.159	can2	TxCan2	8	00000112 09 02 03 04 05 06 07 08	
411	3:57:32.160	can2	netflt4	8	00000212 09 02 03 04 05 06 07 83	
412	3:57:32.160	can2	TxCan2	8	00000112 0A 02 03 04 05 06 07 08	
413	3:57:32.161	can2	netflt4	8	00000212 0A 02 03 04 05 06 07 83	
414	3:57:32.162	can2	TxCan2	8	00000112 0B 02 03 04 05 06 07 08	
415	3:57:32.163	can2	netflt4	8	00000212 0B 02 03 04 05 06 07 83	
416	3:57:32.164	can2	TxCan2	8	00000112 0C 02 03 04 05 06 07 08	
417	3:57:32.165	can2	netflt4	8	00000212 0C 02 03 04 05 06 07 83	
418	3:57:32.166	can2	TxCan2	8	00000112 0D 02 03 04 05 06 07 08	
419	3:57:32.167	can2	netflt4	8	00000212 0D 02 03 04 05 06 07 83	
420	3:57:32.167	can2	TxCan2	8	00000112 0E 02 03 04 05 06 07 08	
421	3:57:32.168	can2	netflt4	8	00000212 0E 02 03 04 05 06 07 83	
422	3:57:32.169	can2	TxCan2	8	00000112 0F 02 03 04 05 06 07 08	
423	3:57:32.169	can2	netflt4	8	00000212 0F 02 03 04 05 06 07 83	
424	3:57:32.170	ind			Event42 RTC: 14:01:25: Date: ...	Start Step16

Step 16: Xtd CAN2 Network Filter4 Test

Box1 user message26: can1->0000113 00 02 03 04 05 06 07 08
Box1 can2flt6: can1->0000213 -- -- -- -- -- -- -- --
Box1 sends can1->0000113 NN 00 00 00 00 00 00 00
Box2 responds with can1->0000213 NN 00 00 00 00 00 00 00 (can2flt6) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

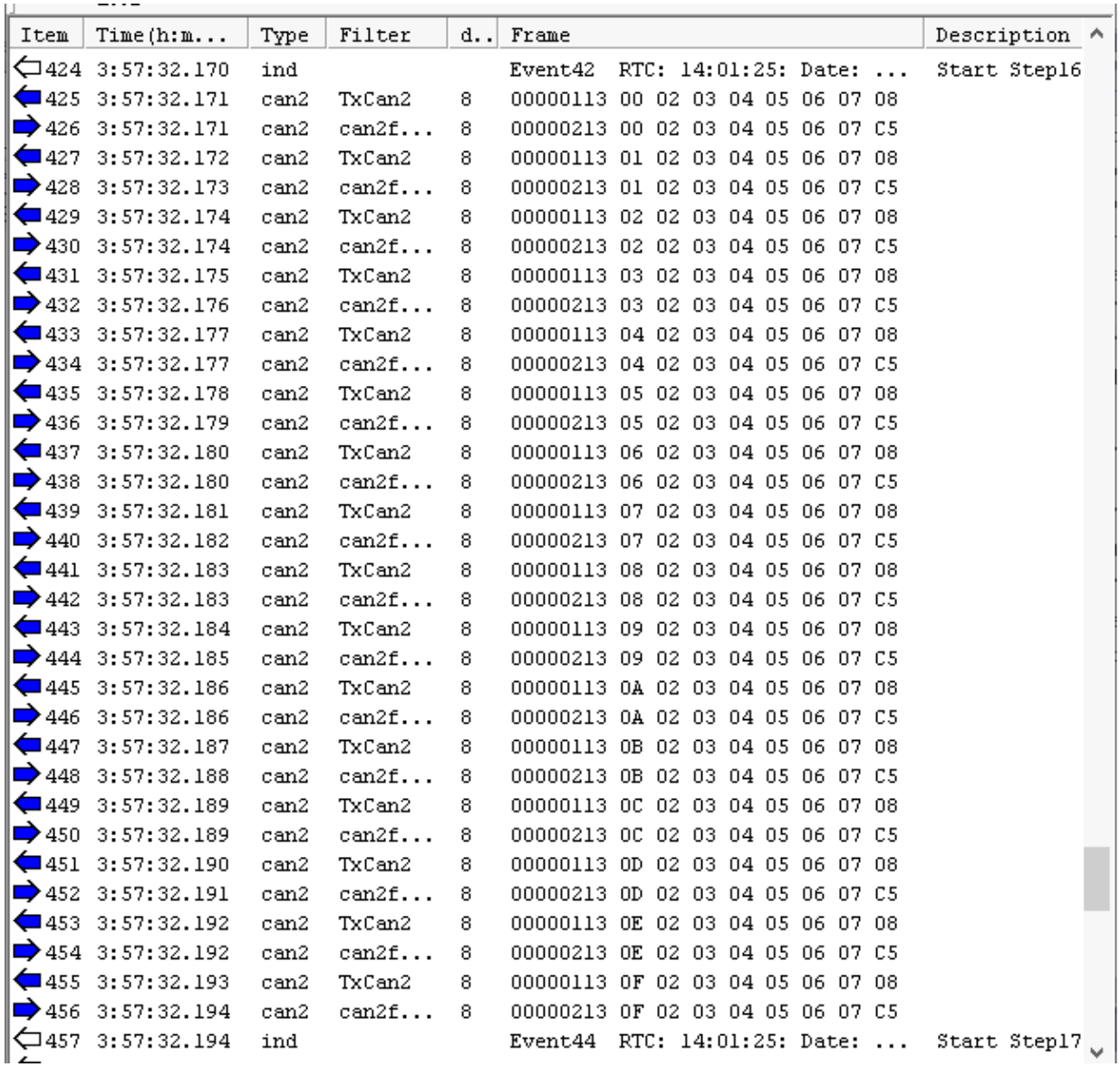
Event42: Start Step 16

```
LOGEVNT; //log event
SET x9 = 0h; //initialize count
SET y3 = 0h; //RX messages mask = 0
SET umsgbyte260 = x9; //set first byte of user message26 to x9
QUE usermsg26; //TX user message26
SET timer1 = 100; //allow 100 ms for test
```

Event43: can2flt6 ->

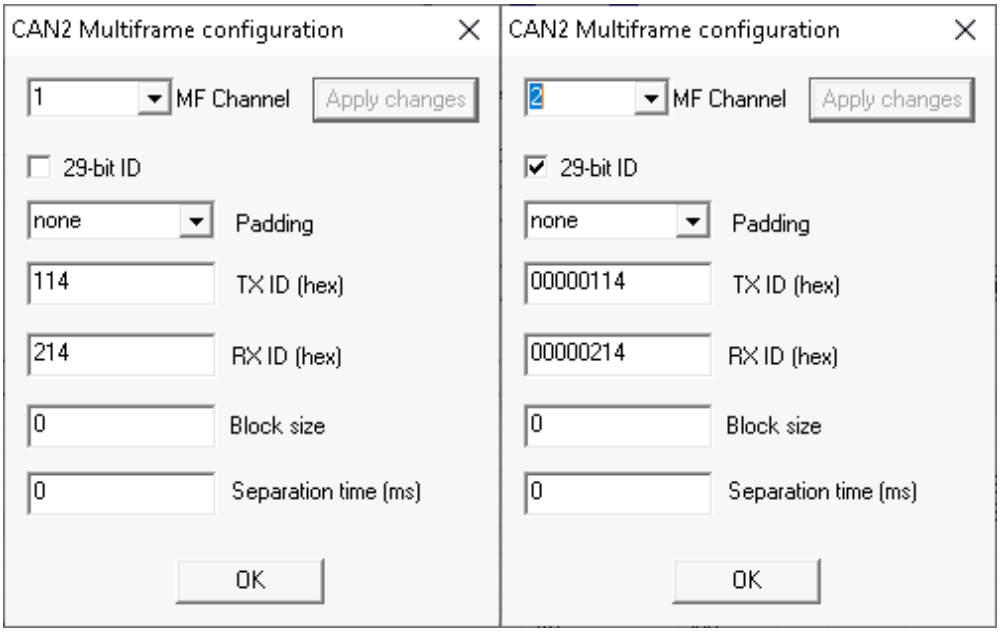
```
SET x1 = rxdata0; //RX first byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask flag for RX message
IF x1 == x9; //if RX byte equals TX byte
IF x9 < Fh; //if not all messages transmitted
    SET x9 + 1h; //increment message count SET umsgbyte260 = x9; //put it
 as first byte in user message26
    QUE usermsg26; //TX user message26
ELSE; //end of test
    IF y3 == FFFFh; //all messages received CL offset+1; //go to next test
 step17
    ELSE; //not all messages received
    CL cl-tag2; //test failed ENDIF; ENDIF;
ENDIF;
```

Test Step 16 Log:



Item	Time(h:m...	Type	Filter	d..	Frame	Description
424	3:57:32.170	ind			Event42 RTC: 14:01:25: Date: ...	Start Step16
425	3:57:32.171	can2	TxCan2	8	00000113 00 02 03 04 05 06 07 08	
426	3:57:32.171	can2	can2f...	8	00000213 00 02 03 04 05 06 07 C5	
427	3:57:32.172	can2	TxCan2	8	00000113 01 02 03 04 05 06 07 08	
428	3:57:32.173	can2	can2f...	8	00000213 01 02 03 04 05 06 07 C5	
429	3:57:32.174	can2	TxCan2	8	00000113 02 02 03 04 05 06 07 08	
430	3:57:32.174	can2	can2f...	8	00000213 02 02 03 04 05 06 07 C5	
431	3:57:32.175	can2	TxCan2	8	00000113 03 02 03 04 05 06 07 08	
432	3:57:32.176	can2	can2f...	8	00000213 03 02 03 04 05 06 07 C5	
433	3:57:32.177	can2	TxCan2	8	00000113 04 02 03 04 05 06 07 08	
434	3:57:32.177	can2	can2f...	8	00000213 04 02 03 04 05 06 07 C5	
435	3:57:32.178	can2	TxCan2	8	00000113 05 02 03 04 05 06 07 08	
436	3:57:32.179	can2	can2f...	8	00000213 05 02 03 04 05 06 07 C5	
437	3:57:32.180	can2	TxCan2	8	00000113 06 02 03 04 05 06 07 08	
438	3:57:32.180	can2	can2f...	8	00000213 06 02 03 04 05 06 07 C5	
439	3:57:32.181	can2	TxCan2	8	00000113 07 02 03 04 05 06 07 08	
440	3:57:32.182	can2	can2f...	8	00000213 07 02 03 04 05 06 07 C5	
441	3:57:32.183	can2	TxCan2	8	00000113 08 02 03 04 05 06 07 08	
442	3:57:32.183	can2	can2f...	8	00000213 08 02 03 04 05 06 07 C5	
443	3:57:32.184	can2	TxCan2	8	00000113 09 02 03 04 05 06 07 08	
444	3:57:32.185	can2	can2f...	8	00000213 09 02 03 04 05 06 07 C5	
445	3:57:32.186	can2	TxCan2	8	00000113 0A 02 03 04 05 06 07 08	
446	3:57:32.186	can2	can2f...	8	00000213 0A 02 03 04 05 06 07 C5	
447	3:57:32.187	can2	TxCan2	8	00000113 0B 02 03 04 05 06 07 08	
448	3:57:32.188	can2	can2f...	8	00000213 0B 02 03 04 05 06 07 C5	
449	3:57:32.189	can2	TxCan2	8	00000113 0C 02 03 04 05 06 07 08	
450	3:57:32.189	can2	can2f...	8	00000213 0C 02 03 04 05 06 07 C5	
451	3:57:32.190	can2	TxCan2	8	00000113 0D 02 03 04 05 06 07 08	
452	3:57:32.191	can2	can2f...	8	00000213 0D 02 03 04 05 06 07 C5	
453	3:57:32.192	can2	TxCan2	8	00000113 0E 02 03 04 05 06 07 08	
454	3:57:32.192	can2	can2f...	8	00000213 0E 02 03 04 05 06 07 C5	
455	3:57:32.193	can2	TxCan2	8	00000113 0F 02 03 04 05 06 07 08	
456	3:57:32.194	can2	can2f...	8	00000213 0F 02 03 04 05 06 07 C5	
457	3:57:32.194	ind			Event44 RTC: 14:01:25: Date: ...	Start Step17

Step 17: Std CAN2 Multi-Frame Test



Multi-frame channel2 configuration for sub-channel 1,2 shown above.
Box1: fill out MF1 64-data bytes from umsgs 8,9,10: can2->114 10 11...4B 4C 4D 4E 4F
and TX MF1 to Box2.
Box2 detects MF1 and send back 64-byte message back: can2->214 -- -- -- -- -- -

Event44: Start Step 17

```
LOGEVNT; // log event
SET timer1 = 100; //allow 100 ms for test
SET TxLong1 = bytesize64; //64-bytes for TX long message SET TxLong1 = usermsg8; //copy data bytes from user messages 8,9,10
SET mfCan102 = TxLong1; //assign TX long to MF1 CAN2 sub-channel 1
SET mfCan102 = bytesize64; //Set MF1 CAN2 to 64 bytes QUE mfCan102; //TX MF1 CAN2 sub-channel 1
```

Event45: mfCanRx102 ->

IF mfCan1 == usermsg8; //if received data equals to content of usermsg8,9,10
CL offset+1; //go to step18
ELSE; //not equals
CL cl-tag4; //test failed
ENDIF;

Test Step 17 Log:

457	3:57:32.194	ind			Event44 RTC: 14:01:25: Date: ... Start Step17	
458	3:57:32.195	can2	TxCan2	8	114 10 40 10 11 12 13 14 15	
459	3:57:32.195	can2	can2flt07	3	214 30 00 00	
460	3:57:32.196	can2	TxCan2	8	114 21 16 17 18 19 1A 1B 1C	
461	3:57:32.197	can2	TxCan2	8	114 22 1D 1E 1F 20 21 22 23	
462	3:57:32.198	can2	TxCan2	8	114 23 24 25 26 27 28 29 2A	
463	3:57:32.199	can2	TxCan2	8	114 24 2B 2C 2D 2E 2F 30 31	
464	3:57:32.200	can2	TxCan2	8	114 25 32 33 34 35 36 37 38	
465	3:57:32.201	can2	TxCan2	8	114 26 39 3A 3B 3C 3D 3E 3F	
466	3:57:32.202	can2	TxCan2	8	114 27 40 41 42 43 44 45 46	
467	3:57:32.203	can2	TxCan2	8	114 28 47 48 49 4A 4B 4C 4D	
468	3:57:32.204	can2	TxCan2	3	114 29 4E 4F	
469	3:57:32.204	can2	can2flt07	8	214 10 40 10 11 12 13 14 15	
470	3:57:32.205	can2	TxCan2	3	114 30 00 00	
471	3:57:32.206	can2	can2flt07	8	214 21 16 17 18 19 1A 1B 1C	
472	3:57:32.207	can2	can2flt07	8	214 22 1D 1E 1F 20 21 22 23	
473	3:57:32.208	can2	can2flt07	8	214 23 24 25 26 27 28 29 2A	
474	3:57:32.209	can2	can2flt07	8	214 24 2B 2C 2D 2E 2F 30 31	
475	3:57:32.210	can2	can2flt07	8	214 25 32 33 34 35 36 37 38	
476	3:57:32.211	can2	can2flt07	8	214 26 39 3A 3B 3C 3D 3E 3F	
477	3:57:32.212	can2	can2flt07	8	214 27 40 41 42 43 44 45 46	
478	3:57:32.213	can2	can2flt07	8	214 28 47 48 49 4A 4B 4C 4D	
479	3:57:32.214	can2	can2flt07	3	214 29 4E 4F	
480	3:57:32.214	ind			Event46 RTC: 14:01:25: Date: ... Start Step18	

Step 18: Xtd CAN2 Multi-Frame Test

Box1: fill out MF1 64-data bytes from umsgs 8,9,10: can2->0000114 10 11...4B 4C 4D 4E 4F
and TX MF1 to Box2.
Box2 detects MF1 and send back 64-byte message back: can2->0000214 -- -- -- -- -- -

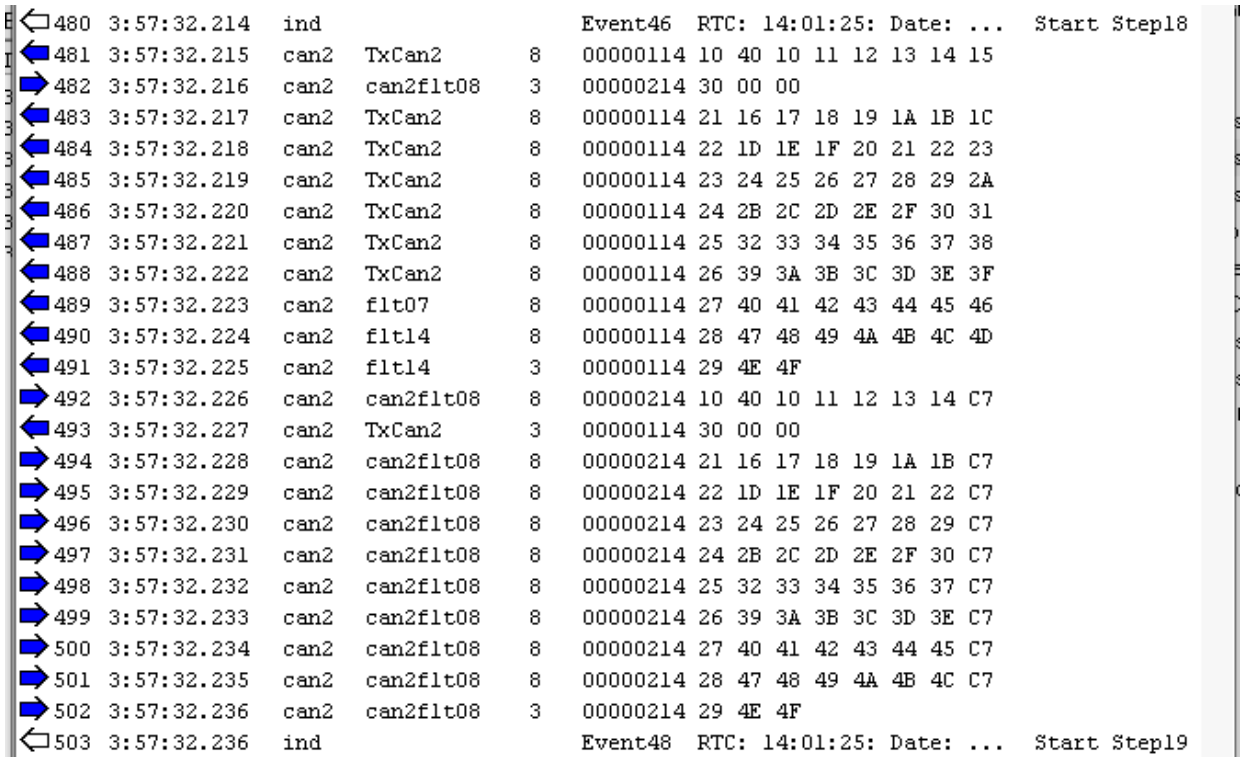
Event46: Start Step 18

```
LOGEVNT; // log event
SET TxLong1 = bytesize64; //64-bytes for TX long message SET TxLong1 = usermsg8; //copy data bytes from user messages 8,9,10
SET mfCan202 = TxLong1; //assign TX long to MF1 CAN2 sub-channel 1
SET mfCan202 = bytesize64; //Set MF1 CAN2 to 64 bytes QUE mfCan202; //TX MF1 CAN2 sub-channel 2
SET timer1 = 100; //allow 100 ms for test
```

Event47: mfCanRx202 ->

```
IF mfCan1 == usermsg8; //if RX multi-frame data match to usermsg8 content IF deviceState18 > 1Dh; //if hardware type >= NW92
    CL offset+1; // go to step 19
ELSE;
    CL cl-tag52; //skip CAN FD tests, go to step 21 (Uart test) ENDIF;
ELSE;
CL cl-tag4; //not match to usermsg8 content
ENDIF;
SET timer1 = 100; // allow100 ms for test
```

Test Step18 Log:



480	3:57:32.214	ind			Event46 RTC: 14:01:25: Date: ... Start Step18	
481	3:57:32.215	can2	TxCan2	8	00000114 10 40 10 11 12 13 14 15	
482	3:57:32.216	can2	can2flt08	3	00000214 30 00 00	
483	3:57:32.217	can2	TxCan2	8	00000114 21 16 17 18 19 1A 1B 1C	
484	3:57:32.218	can2	TxCan2	8	00000114 22 1D 1E 1F 20 21 22 23	
485	3:57:32.219	can2	TxCan2	8	00000114 23 24 25 26 27 28 29 2A	
486	3:57:32.220	can2	TxCan2	8	00000114 24 2B 2C 2D 2E 2F 30 31	
487	3:57:32.221	can2	TxCan2	8	00000114 25 32 33 34 35 36 37 38	
488	3:57:32.222	can2	TxCan2	8	00000114 26 39 3A 3B 3C 3D 3E 3F	
489	3:57:32.223	can2	flt07	8	00000114 27 40 41 42 43 44 45 46	
490	3:57:32.224	can2	flt14	8	00000114 28 47 48 49 4A 4B 4C 4D	
491	3:57:32.225	can2	flt14	3	00000114 29 4E 4F	
492	3:57:32.226	can2	can2flt08	8	00000214 10 40 10 11 12 13 14 C7	
493	3:57:32.227	can2	TxCan2	3	00000114 30 00 00	
494	3:57:32.228	can2	can2flt08	8	00000214 21 16 17 18 19 1A 1B C7	
495	3:57:32.229	can2	can2flt08	8	00000214 22 1D 1E 1F 20 21 22 C7	
496	3:57:32.230	can2	can2flt08	8	00000214 23 24 25 26 27 28 29 C7	
497	3:57:32.231	can2	can2flt08	8	00000214 24 2B 2C 2D 2E 2F 30 C7	
498	3:57:32.232	can2	can2flt08	8	00000214 25 32 33 34 35 36 37 C7	
499	3:57:32.233	can2	can2flt08	8	00000214 26 39 3A 3B 3C 3D 3E C7	
500	3:57:32.234	can2	can2flt08	8	00000214 27 40 41 42 43 44 45 C7	
501	3:57:32.235	can2	can2flt08	8	00000214 28 47 48 49 4A 4B 4C C7	
502	3:57:32.236	can2	can2flt08	3	00000214 29 4E 4F	
503	3:57:32.236	ind			Event48 RTC: 14:01:25: Date: ... Start Step19	

Step 19: Std CAN2 FD Test

Box1: TX usermsg27 (CAN FD) can2->115 00 01 ...3E 3F (64 bytes) Box2: Responds with (CAN FD) can2->215 00 01...3E 3F (64 bytes) can2flt14 Box1 verifies selected RX bytes and if match test pass

Event48: Start Step 19

```
LOGEVNT; //log event
IF deviceState18 > 1Dh; //if hardware is NW92 and above QUE usermsg27;
//TX CAN2 FD message
SET timer1 = 100; //allow 100 ms for test
ELSE; //if not CAN FD hardware
CL cl-tag2; //test failed
ENDIF;
```

Event49: can2flt14 ->

```
IF rxdata15 != Fh; //verify RX byte 16
SET x1 = Fh;
SET x2 = rxdata15; CL cl-tag5; //byte 16 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata39 != 27h; //verify RX byte 40
SET x1 = 27h;
SET x2 = rxdata39; CL cl-tag5; //byte 40 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata63 != 3Fh; //verify RX byte 64
SET x1 = 3Fh;
SET x2 = rxdata63; CL cl-tag5; //byte 64 not valid, test failed
ENDCL; //stop verification
ENDIF;
CL offset+1; //Ok, go to step 20
```

Test Step 19, 20 Log:

```
502 3...  can2  can2flt06    3   00000214 29 4E 4F
503 3...  ind                   Event48  RTC: 14:01:25: Date: 03/19/2022                                  Start Step19 CAN2 FD ...
504 3...  can2  TxCan2       16  115 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
505                          24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
506                          24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
507 3...  can2  can2flt14    16  215 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
508                          24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
509                          24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
510 3...  ind                   Event50  RTC: 14:01:25: Date: 03/19/2022                                  Start Step20 CAN2 FD ...
511 3...  can2  TxCan2       16  00000115 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
512                          24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
513                          24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
514 3...  can2  can2flt15    16  00000215 00 01 02 03 04 05 06 CE 08 09 0A 0B 0C 0D 0E 0F
515                          24  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
516                          24  28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
517 3...  ind                   Event52  RTC: 14:01:25: Date: 03/19/2022                                  Start Step21 UBP1 Use...
```

Step 20: Xtd CAN2 FD Test

Box1: TX usermsg27 (CAN FD) can2->115 00 01 ...3E 3F (64 bytes) Box2: Responds with (CAN FD) can2->215 00 01...3E 3F (64 bytes) can2flt14 Box1 verifies selected RX bytes and if match test pass

Event50: Start Step 20

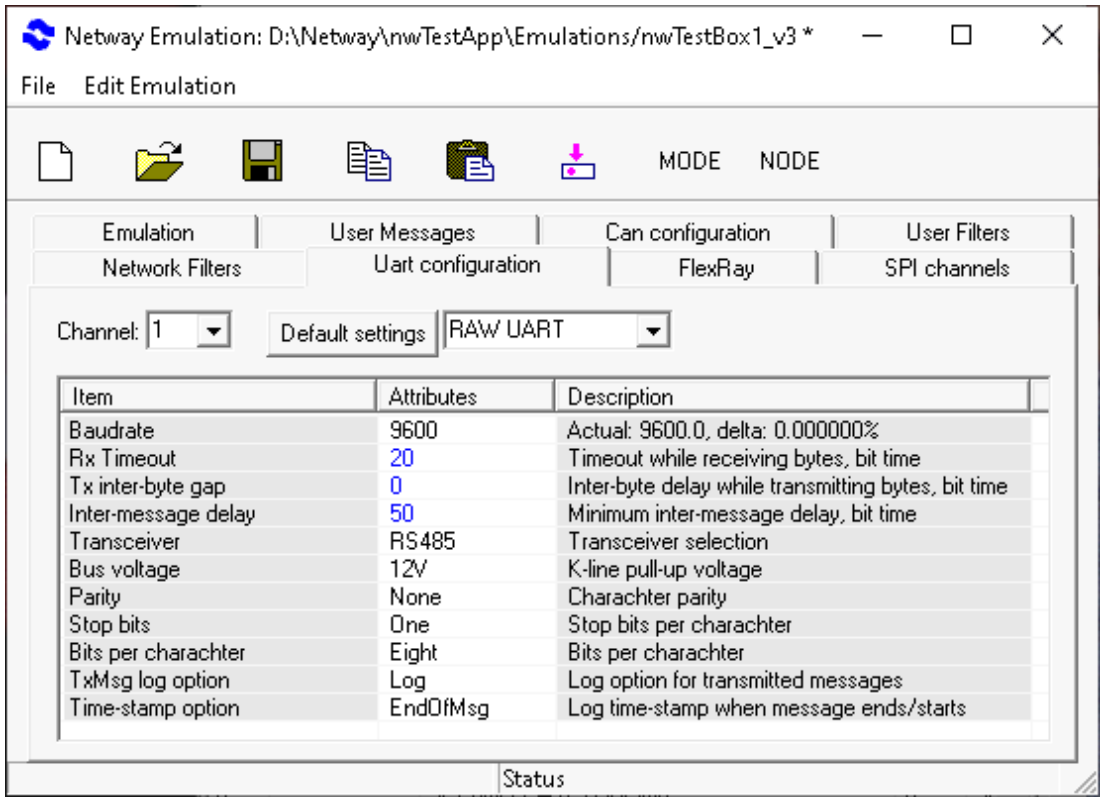
```
SET x1 = deviceState18; //hardware type in x1
IF deviceState18 > 1Dh; //if hardware is NW92 and above SET timer1 =
100; //allow 100 ms for test
QUE usermsg31; //TX CAN2 FD user message31
LOGEVNT; //log event
ELSE; //if not CAN FD hardware
CL cl-tag8; //test failed
ENDIF;
```

Event51: can2flt15 ->

```
IF rxdata15 != Fh; //verify RX byte 16
SET x1 = Fh;
SET x2 = rxdata15; CL cl-tag5; //byte 16 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata39 != 27h; //verify RX byte 40
SET x1 = 27h;
SET x2 = rxdata39; CL cl-tag5; //byte 40 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata63 != 3Fh; //verify RX byte 64
SET x1 = 3Fh;
SET x2 = rxdata63; CL cl-tag5; //byte 64 not valid, test failed
ENDCL; //stop verification
ENDIF;
CL cl-tag52; //OK, go to step 21
```

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

UBP1 Test Steps



Uart configuration for channel 1 is shown above.
Raw Uart protocols is a protocol without rules, no checksum, end of frame is defined by RX time-out.

Step 21: UBP1 UserFlt/NetFlt Test

Box1: sending user message 41 ubp1-> 11 xx 01 02 03 04 05 06 07
and user message 42 ubp1-> 12 xx 01 02 03 04 05 06 07
Receives ubp1->21 xx (filter5) and ubp1->22 xx (netflt5)

Event53: Start Step 21

```
LOGEVNT; //log event
SET x9 = 0h; //initialize count for user filter5
SET x10 = 0h; //initialize count for netflt5
SET x11 = 0h;
SET x12 = 0h;
IF protocol4 == 1h; //if Uart1 is available
SET umsgbyte411 = x9; //put count to first byte of user message41
SET umsgbyte421 = x9; //put count to first byte of user message42
QUE usermsg41; //TX user message41 for user filters
QUE usermsg42; //TX user message42 for network filters
ENDIF;
SET timer1 = 1200; //allow 1.2 second for test
```

Event54: filter5 -> (ubp1->21 -- -- --)

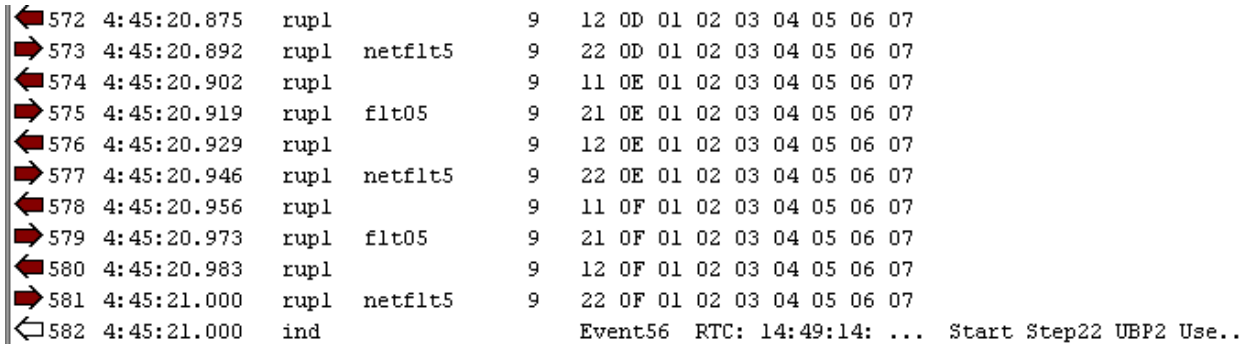
```
SET x1 = rxdata1; //second RX byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask for RX message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 15, continue TX user message41
    SET x9 + 1h; //increment count SET umsgbyte411 = x9; //put count to
 second byte of user message41
    QUE usermsg41; //TX user message41
ENDIF;
ELSE;
CL cl-tag9; //test failed, data mismatch
ENDIF;
```

Event55: netflt5 -> (ubp1->22 -- -- --)

```
SET x1 = rxdata1; //second RX byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask for RX message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 15, continue TX user message41
    SET x9 + 1h; //increment count SET umsgbyte421 = x10; //put count to
 second byte of user message42
    QUE usermsg42; //TX user message42
ELSE;
    CL offset+1; //success, go to step 22
    ENDCL; ENDIF;
ELSE;
CL cl-tag9; //test failed, data mismatch
ENDIF;
```

Test Step 21 Log:

```
516                                          24  28 29 2A 2B 2C 2D 2E 2F ...
517  4:45:20.149   ind                           Event52  RTC: 14:49:13: ...  Start Step21 UBP1
518  4:45:20.158   rupl                9   11 00 01 02 03 04 05 06 07
519  4:45:20.175   rupl   flt05        9   21 00 01 02 03 04 05 06 07
520  4:45:20.185   rupl                9   12 00 01 02 03 04 05 06 07
521  4:45:20.202   rupl   netflt5      9   22 00 01 02 03 04 05 06 07
522  4:45:20.211   rupl                9   11 01 01 02 03 04 05 06 07
523  4:45:20.228   rupl   flt05        9   21 01 01 02 03 04 05 06 07
524  4:45:20.238   rupl                9   12 01 01 02 03 04 05 06 07
525  4:45:20.255   rupl   netflt5      9   22 01 01 02 03 04 05 06 07
526  4:45:20.264   rupl                9   11 02 01 02 03 04 05 06 07
527  4:45:20.281   rupl   flt05        9   21 02 01 02 03 04 05 06 07
528  4:45:20.290   rupl                9   12 02 01 02 03 04 05 06 07
529  4:45:20.307   rupl   netflt5      9   22 02 01 02 03 04 05 06 07
```



```
572  4:45:20.875   rupl                9   12 0D 01 02 03 04 05 06 07
573  4:45:20.892   rupl   netflt5      9   22 0D 01 02 03 04 05 06 07
574  4:45:20.902   rupl                9   11 0E 01 02 03 04 05 06 07
575  4:45:20.919   rupl   flt05        9   21 0E 01 02 03 04 05 06 07
576  4:45:20.929   rupl                9   12 0E 01 02 03 04 05 06 07
577  4:45:20.946   rupl   netflt5      9   22 0E 01 02 03 04 05 06 07
578  4:45:20.956   rupl                9   11 0F 01 02 03 04 05 06 07
579  4:45:20.973   rupl   flt05        9   21 0F 01 02 03 04 05 06 07
580  4:45:20.983   rupl                9   12 0F 01 02 03 04 05 06 07
581  4:45:21.000   rupl   netflt5      9   22 0F 01 02 03 04 05 06 07
582  4:45:21.000   ind                     Event56  RTC: 14:49:14: ...  Start Step22 UBP2 Use..
```

Step 22: UBP2 UserFlt/NetFlt Test

Box1: sending user message 43 ubp2-> 11 xx 01 02 03 04 05 06 07
and user message 44 ubp2-> 12 xx 01 02 03 04 05 06 07
Receives ubp2->21 xx (filter6) and ubp2->22 xx (netflt6)

Event56: Start Step 22

```
IF protocol14 == 1h; //if Uart2 is available, start test step 22
LOGEVNT; //log event
SET x9 = 0h; //reset count
SET umsgbyte431 = x9; //put count to second byte of user message43
SET umsgbyte441 = x9; //put count to second byte of user message44
QUE usermsg43; //TX user message43
QUE usermsg44; //TX user message44
ELSE; //Uart2 is not available
CL cl-tag60; //go to step 23
ENDIF;
SET timer1 = 1000; //allow 1 seconds for test
```

Event57: filter6 -> (ubp2->21 -- -- --)

```
SET x1 = rxdata1; //second RX byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask for RX message
IF x1 == x11; //if RX byte equals to count
IF x11 < Fh; //if less than 15, continue TX user message41
    SET x11 + 1h; //increment count SET umsgbyte431 = x11; //put count to
 second byte of user message43
    QUE usermsg43; //TX user message43
ENDIF;
ELSE; //RX byte not equal count
CL cl-tag9; //test failed, data mismatch
ENDIF;
```

Event58: netflt6 -> (ubp2->22 -- -- --)

```
SET x1 = rxdata1; //second RX byte to x1
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1; SET y3 | y2; //set mask for RX message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 15, continue TX user message41
    SET x9 + 1h; //increment count SET umsgbyte421 = x10; //put count to
 second byte of user message42
    QUE usermsg42; //TX user message42
ELSE;
    CL offset+1; //verify step 21,22 results ENDCL; ENDIF;
ELSE;
CL cl-tag9; //test failed, data mismatch
ENDIF;
```

Event59: Verify Steps 21, 22 Results

```
SET timer1 = 0; IF y3 != FFFFh; //if not all filter5 messages received
CL cl-tag6; //test failed
ENDCL;
ENDIF;
IF y4 != FFFFh; //if not all netflt5 messages received
CL cl-tag6; //test failed
ENDCL;
ENDIF;
IF y5 != FFFFh; //if not all filter6 messages received
CL cl-tag6; //test failed
ENDCL;
ENDIF;
IF y6 != FFFFh; //if not all netflt5 messages received
CL cl-tag6; //test failed
ENDCL;
ENDIF;
CL offset+1; //go to step 23
```

Test Step 22 Log:

```
582  4:45:21.000   ind                      Event56  RTC: 14:49:14: ...  Start Step22 UBP2
583  4:45:21.005   rup2                9    11 00 01 02 03 04 05 06 07
584  4:45:21.013   rup2   flt06        9    21 00 01 02 03 04 05 06 07
585  4:45:21.019   rup2                9    12 00 01 02 03 04 05 06 07
586  4:45:21.027   rup2   netflt6      9    22 00 01 02 03 04 05 06 07
587  4:45:21.033   rup2                9    11 01 01 02 03 04 05 06 07
588  4:45:21.041   rup2   flt06        9    21 01 01 02 03 04 05 06 07
589  4:45:21.047   rup2                9    12 01 01 02 03 04 05 06 07
590  4:45:21.055   rup2   netflt6      9    22 01 01 02 03 04 05 06 07
591  4:45:21.060   rup2                9    11 02 01 02 03 04 05 06 07
592  4:45:21.069   rup2   flt06        9    21 02 01 02 03 04 05 06 07
593  4:45:21.074   rup2                9    12 02 01 02 03 04 05 06 07
594  4:45:21.083   rup2   netflt6      9    22 02 01 02 03 04 05 06 07
595  4:45:21.088   rup2                9    11 03 01 02 03 04 05 06 07
```

```
636  4:45:21.377   rup2   flt06        9    21 0D 01 02 03 04 05 06 07
637  4:45:21.383   rup2                9    12 0D 01 02 03 04 05 06 07
638  4:45:21.391   rup2   netflt6      9    22 0D 01 02 03 04 05 06 07
639  4:45:21.396   rup2                9    11 0E 01 02 03 04 05 06 07
640  4:45:21.405   rup2   flt06        9    21 0E 01 02 03 04 05 06 07
641  4:45:21.410   rup2                9    12 0E 01 02 03 04 05 06 07
642  4:45:21.419   rup2   netflt6      9    22 0E 01 02 03 04 05 06 07
643  4:45:21.424   rup2                9    11 0F 01 02 03 04 05 06 07
644  4:45:21.433   rup2   flt06        9    21 0F 01 02 03 04 05 06 07
645  4:45:21.439   rup2                9    12 0F 01 02 03 04 05 06 07
646  4:45:21.447   rup2   netflt6      9    22 0F 01 02 03 04 05 06 07
647  4:45:21.448   ind                      Event60  RTC: 14:49:14: ...  Start Step 23
```

Step 23: UBP1/2 Long Uart Test

User message36 84-byte message is configured as follows: ubp1-> 15 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14
15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52

Event60: Start Step 23

```
LOGEVNT; //log event
SET timer1 = 2000; //allow 2 seconds for test
QUE usermsg36; //TX user message36
```

Event61: ubpLonRx1->

We received long Uart1 message in response to user message 36:

```
SET x1 = RxLong1; //copy all RX bytes to x1...
SET x2 = RxLongLen1; //length of RX long message to x2
IF x1 == 25h; //first byte of RX long message equals 25h
SET umsgbyte80 = 13h; //put 13h to first byte of user mesage8
SET TxLong1 = bytesize64; //assign message length SET TxLong1 =
protocol4; //assign message protocol/channel Uart1
SET TxLong1 = usermsg8; //copy Long message bytes from usermsg8
QUE TxLong1; //TX Long message pre-configured above
SET umsgbyte80 = 10h; //restore first byte of user message8
ELSEIF x1 == 23h; //first byte of RX long message equals 23h
IF x2 == 40h; // if RX length equals 64
    IF protocol14 == 1; //if ubp2 available QUE usermsg46; //TX long user
 message46 on Uart2
    ELSE; CL cl-tag64; //Uart2 is not available, go to step 24
    ENDIF; ENDIF;
ENDIF;
```

Event62: ubpLonRx2->

We received long Uart2 message in response to user message 46:

```
SET x1 = RxLong2; //copy all RX bytes to x1...
SET x2 = RxLongLen2; //length of RX long message to x2
IF x1 == 25h; //first byte of RX long message equals 25h
SET umsgbyte80 = 13h; //set first byte of user message8 to 13h SET
TxLong2 = bytesize64; //assign length of TX long for channel2
SET TxLong2 = protocol14; //assign protocol/channel to Uart2
SET TxLong2 = usermsg8; //copy all bytes from user message8 to TX long
QUE TxLong2; //TX long on Uart2
SET umsgbyte80 = 10h; //restore first byte of user message8
ELSEIF x1 == 23h; //first byte of RX long message equals 23h
IF x2 == 40h; //if RX length equals 64
    CL cl-tag64; //go to step 24
ENDIF;
ENDIF;
```

Test Step 23 Log:

```
647 4:45:21.448  ind          Event60  RTC: 14:49:14: Date: 03/19/2022                        Start Step 23 Long ub.
648 4:45:21.537  rupl     18  15 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
649                       22  11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26
650                       22  27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C
651                       22  3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52
652 4:45:21.633  rupl     18  25 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
653                       22  11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26
654                       22  27 28 29 2A 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
655                       22  38 39 3A 3B 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52
656 4:45:21.702  rupl     18  13 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
657                       22  22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
658                       22  38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D
659                       2   4E 4F
660 4:45:21.777  rupl     18  23 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
661                       22  22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
662                       22  38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D
663                       2   4E 4F
664 4:45:21.823  rup2     18  15 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
665                       22  11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26
666                       22  27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C
667                       22  3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52
668 4:45:21.864  rup2     18  25 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
669                       22  11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26
670                       22  27 28 29 2A 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
671                       8   38 39 3A 3B 3C 3D 3E 3F
672 4:45:21.899  rup2     18  13 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
673                       22  22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
674                       22  38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D
675                       2   4E 4F
676 4:45:21.937  rup2     18  23 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
677                       22  22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
678                       22  38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D
679                       2   4E 4F
680 4:45:21.937  ind          Event64  RTC: 14:49:15: Date: 03/19/2022                        Start Step 24 - mixed.
```

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

Mix Protocols CAN1/2+UBP1/2

Test mix protocols at the same time

Step 24: Mixed Protocols CAN1/2, UBP1/2

Box1 TX user message 51,52,53,54 and receives responses from Box2 as netflt7,8,9,10
Box1 TX user message:
user message51: can1-> 116 XX1 02 03 04 05 06 07
user message52: can2-> 116 XX 01 02 03 04 05 06 07
user message53: ubp1-> 16 XX 01 02 03 04 05 06 07
user message54: ubp2-> 16 XX 01 02 03 04 05 06 07
Box2 responses:
netflt7: can1-> 216 XX -- -- -- -- -- -- --
netflt8: can1-> 216 XX -- -- -- -- -- -- --
netflt9: ubp1-> 26 XX -- -- -- -- -- -- --
netflt10: ubp2-> 26 XX -- -- -- -- -- -- --

Event64: Start Step 24

```
LOGEVNT; //log event
SET x9 = 0h; //initialize counts x9-x12
SET x10 = 0h; SET x11 = 0h; SET x12 = 0h; SET y3 = 0h; //initialize masks y3-y6
SET y4 = 0h; SET y5 = 0h; SET y6 = 0h; SET umsgbyte510 = x9; //set first byte of user message51 with count QUE usermsg51; //TX user message51
SET umsgbyte520 = x10; //set first byte of user message52 with count QUE usermsg52; //TX user message52
SET umsgbyte531 = x11; //set first byte of user message53 with count QUE usermsg53; //TX user message53
IF protocol14 == 1; //if ubp2 is available
SET umsgbyte541 = x12; //set first byte of user message54 with count QUE usermsg54; //TX user message54
ENDIF;
SET timer1 = 1000; //allow 1 second for test
```

Event65: netflt7->

```
SET x1 = rxdata0; //first RX byte to x1
```

```
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for
received message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 16 messages received
    SET x9 + 1h; //increment count SET umsgbyte510 = x9; //set first byte to
 count QUE usermsg51; //continue TX user message51
ELSE; //all 16 messages have been sent
    IF y3 != FFFFh; //if not all messages received CL cl-tag2; //test failed
 ENDIF; ENDIF;
ELSE; //RX byte not equals to count, error
CL cl-tag9; //data mismatch, test failed
ENDIF;
```

Event66: netflt8->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for
received message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 16 messages received
    SET x9 + 1h; //increment count SET umsgbyte520 = x9; //set first byte to
 count QUE usermsg52; //continue TX user message52
ELSE; //all 16 messages have been sent
    IF y3 != FFFFh; //if not all messages received CL cl-tag2; //test failed
 ENDIF; ENDIF;
ELSE; //RX byte not equals to count, error
CL cl-tag9; //data mismatch, test failed
ENDIF;
```

Event65: netflt9->

Uart1 is slowest protocol, go to next step when 16 messages received!

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for
received message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 16 messages received
```

```
    SET x9 + 1h; //increment count SET umsgbyte530 = x9; //set first byte to
 count QUE usermsg53; //continue TX user message53
ELSE; //all 16 messages have been sent
    IF y3 != FFFFh; //if not all messages received CL cl-tag2; //test failed
 ELSE; CL cl-tag70; //go to next step 25
    ENDIF; ENDIF;
ELSE; //RX byte not equals to count, error
CL cl-tag9; //data mismatch, test failed
ENDIF;
```

Event65: netflt10->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for
received message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 16 messages received
    SET x9 + 1h; //increment count SET umsgbyte540 = x9; //set first byte to
 count QUE usermsg54; //continue TX user message54
ELSE; //all 16 messages have been sent
    IF y3 != FFFFh; //if not all messages received CL cl-tag2; //test failed
 ENDIF; ENDIF;
ELSE; //RX byte not equals to count, error
CL cl-tag9; //data mismatch, test failed
ENDIF;
```

Test Step 24 Log:

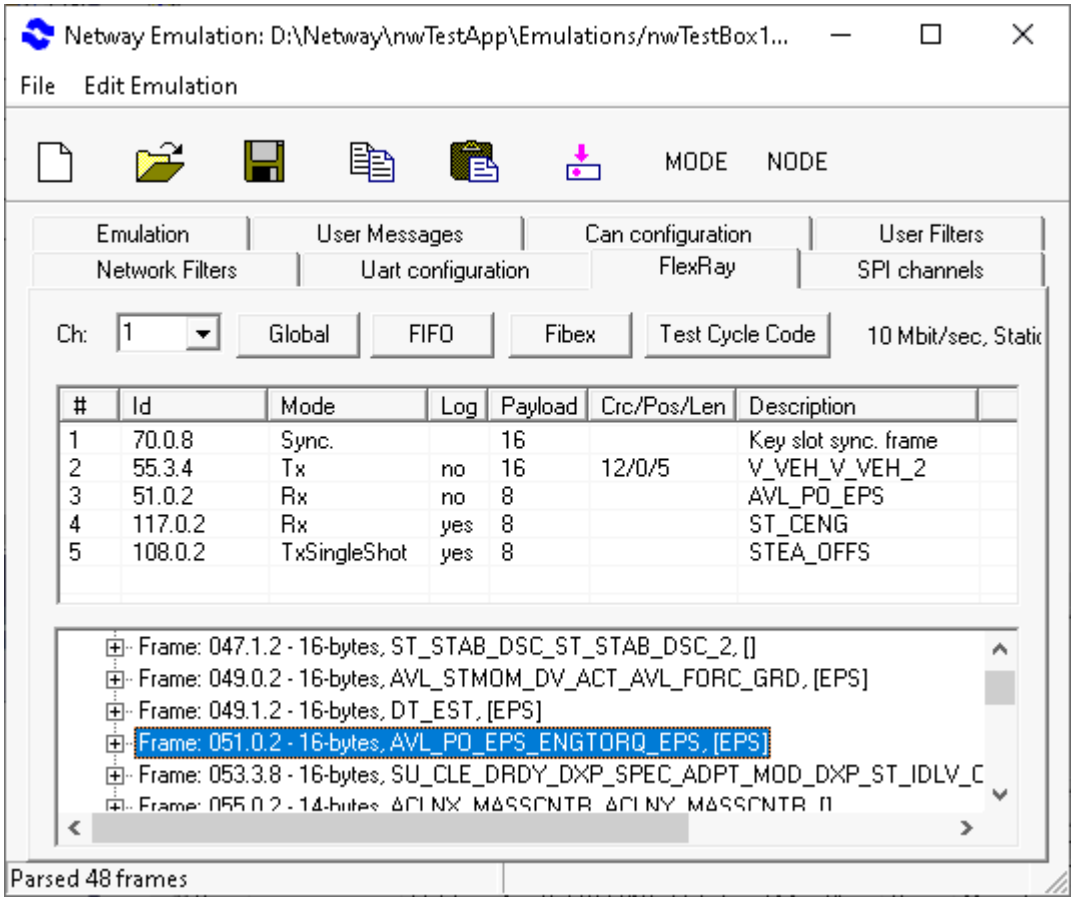
```
680 4:45:21.937  ind                  Event64  RTC: 14:49:15: D...   Start Step 24 - mixed ...
681 4:45:21.938  can1   TxCan1     8   116 00 01 02 03 04 05 06 07
682 4:45:21.938  can2   TxCan2     8   116 00 01 02 03 04 05 06 07
683 4:45:21.939  can1   netflt7    8   216 00 00 00 00 00 00 00 00
684 4:45:21.939  can2   netflt8    8   216 00 00 00 00 00 00 00 00
685 4:45:21.940  can1   TxCan1     8   116 01 01 02 03 04 05 06 07
686 4:45:21.940  can2   TxCan2     8   116 01 01 02 03 04 05 06 07
687 4:45:21.940  can1   netflt7    8   216 01 00 00 00 00 00 00 00
688 4:45:21.941  can2   netflt8    8   216 01 00 00 00 00 00 00 00
689 4:45:21.942  can1   TxCan1     8   116 02 01 02 03 04 05 06 07
690 4:45:21.942  can1   netflt7    8   216 02 00 00 00 00 00 00 00
691 4:45:21.942  can2   TxCan2     8   116 02 01 02 03 04 05 06 07
692 4:45:21.943  can2   netflt8    8   216 02 00 00 00 00 00 00 00
693 4:45:21.942  rup2               9   16 00 01 02 03 04 05 06 07
694 4:45:21.943  can1   TxCan1     8   116 03 01 02 03 04 05 06 07
695 4:45:21.944  can1   netflt7    8   216 03 00 00 00 00 00 00 00
696 4:45:21.944  can2   TxCan2     8   116 03 01 02 03 04 05 06 07
697 4:45:21.945  can1   TxCan1     8   116 04 01 02 03 04 05 06 07
698 4:45:21.945  can2   netflt8    8   216 03 00 00 00 00 00 00 00
```

```
798 4:45:22.228  rup1   netflt9    9   26 0A 01 02 03 04 05 06 07
799 4:45:22.238  rup1               9   16 0B 01 02 03 04 05 06 07
800 4:45:22.254  rup1   netflt9    9   26 0B 01 02 03 04 05 06 07
801 4:45:22.265  rup1               9   16 0C 01 02 03 04 05 06 07
802 4:45:22.281  rup1   netflt9    9   26 0C 01 02 03 04 05 06 07
803 4:45:22.291  rup1               9   16 0D 01 02 03 04 05 06 07
804 4:45:22.307  rup1   netflt9    9   26 0D 01 02 03 04 05 06 07
805 4:45:22.317  rup1               9   16 0E 01 02 03 04 05 06 07
806 4:45:22.333  rup1   netflt9    9   26 0E 01 02 03 04 05 06 07
807 4:45:22.344  rup1               9   16 0F 01 02 03 04 05 06 07
808 4:45:22.360  rup1   netflt9    9   26 0F 01 02 03 04 05 06 07
809 4:45:22.360  ind                Event71  RTC: 14:49:15: D...   Start Step 25 Flexray
```

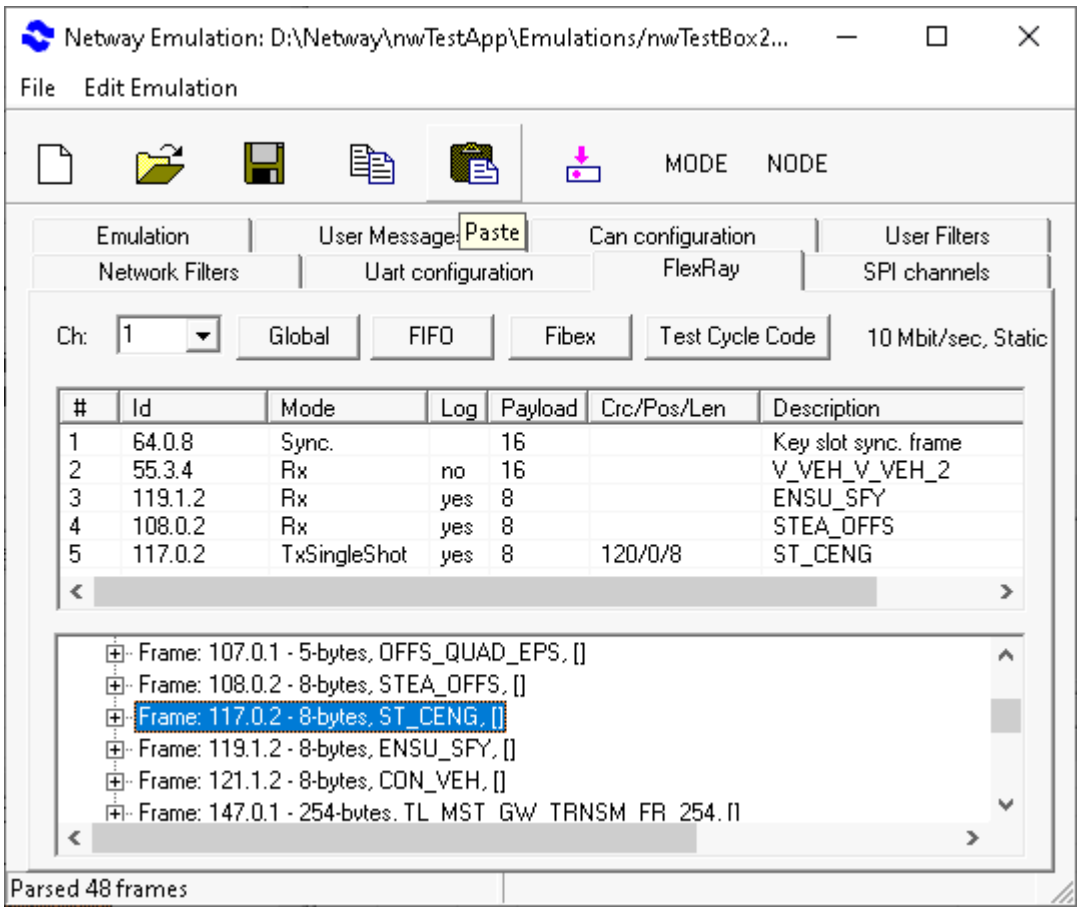
Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

Flexray Test

Box 1 Flexray configuration is shown below:



Box 2 Flexray configuration is shown below:



Notice Key slot sync frames for both Boxes are imported from Fibex.
Minimum two sync Nodes are needed to start normal communications.
Box1 transmits slot 55 and receives slots 51, 117, 108. Box2 transmits slot 117 and receives slots 55,119,108.
Box1 configured logging of slot 117, 108, Box2 - logging slots 119,108,117
Static slots (ID less than 91) are constantly transmitted, they are not set for log, but they still able to trigger event and data can be processed per emulation.

Step 25: Flexray Test

User message56 (slot107) fray1->06C 11 22 XX 44 55 66 77 88 (XX range 00 -: 0F)
Slot55-static slot transmitted periodically by Box1
User message58 (slot55) fray1->037 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
User message59 (slot55) fray1->037 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Slot 117 has CRC protected PDU - two first bytes are used for CRC and keep alive count.
We are going to use bytes starting from byte 3 (index2)
Box1 transmits 16 frames slot 108 and in response receives 16 frames 117 from Box2
After 16 slot107 transmitted by Box1 and 16 slot117 is received and count confirmed,
Box1 transmits user message58 to change slot55 from default to 00-:-0F
Box2 detects RX slot55 with non-default values and copies 6 bytes to slot117 signaling Box1 to finish test

Event71: Start Step 25

```
IF protocol21 == 1; //if Flexray protocol is enabled
LOGEVNT; //log event
SET umsgbyte600 = 21; //signal start of fray test to Box2
QUE usermsg60; //TX CAN->101 15...(15h = 21-fray protocol)
QUE usermsg59; //slot55 to default values (0xFF) TX user message59
SET delay2 = 50; //delay further commands 50ms to ensure default values RX by Box2
SET x9 = 0h; //initialize counter
SET y3 = 0h; //initialize RX messages mask
SET umsgbyte562 = x9; //set third byte of user message56 - slot 108
QUE usermsg56; //TX user message56
SET timer1 = 500; //allow 500 ms for test
ELSE; //if no Flexray
CL cl-tag74; //go to next step
```

```
ENDIF;
```

Event72: frayFrame104 (ch1, frame4, slot 117)

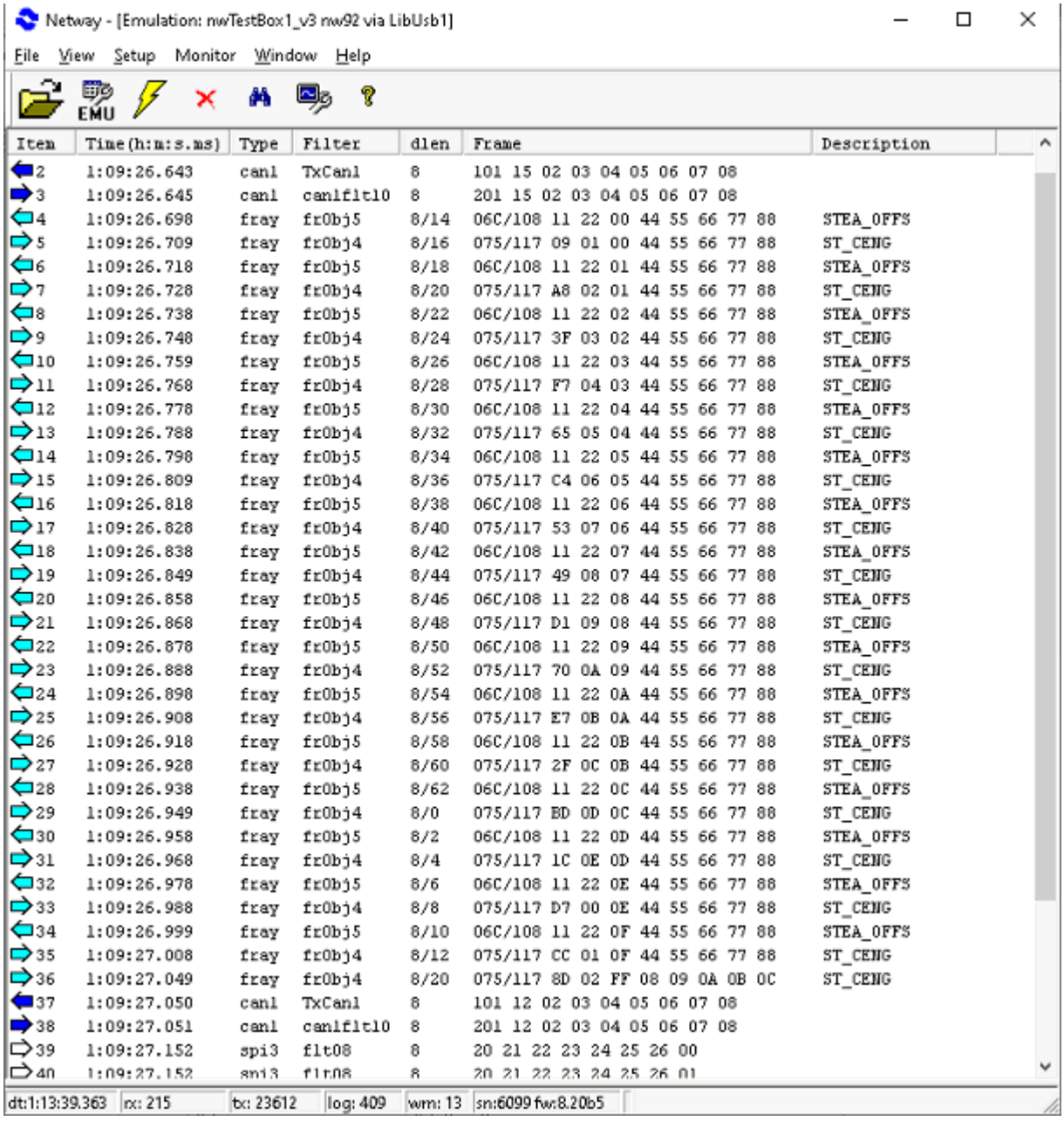
```
ENABLE bytesize6; //copy size - 6 bytes
SET x1 = rxdata2; //6 bytes starting from byte 2 to x1-6
IF x1 == FFh; //if initial value
ENDCL; //end of this command line
ENDIF;
SET y1 = x1;
SET y2 = 1h;
SET y2 << y1;
SET y3 | y2; //set mask for received message
IF x1 == x9; //if RX byte equals to count
IF x9 < Fh; //if less than 16 messages are received
    SET x9 + 1h; //increment count SET umsgbyte562 = x9; //second byte of
 user message56(slot108)= x9
    QUE usermsg56; //TX user message56 with new count ELSE; //all
 messages have been received
    IF y3 == FFFFh; //all messages have been received SET timer1 = 0;
 //time-out off
    QUE usermsg58; //change slot 55 data
    ELSE;
    CL cl-tag6;
    ENDIF;
ENDIF;
ENDIF;
```

Event74: Verify Test Results

```
//x1-x6 prepared in event 72
IF x2 == 8h; //compare RX bytes to expected (8,9,A,B,C)
IF x3 == 9h;
    IF x4 == Ah;
    IF x5 == Bh;
    IF x6 == Ch;
    IF x87 == 0; //x87 - do it once flag
    SET x87 = 87h;
```

```
    QUE usermsg59; //slot 55 to default values (0xFF)
    CL cl-tag74; //go to next step
    ENDCL;
    ENDIF;
    ENDIF;
    ENDIF;
    ENDIF;
ENDIF;
ENDIF;
```

Flexray Test Log:



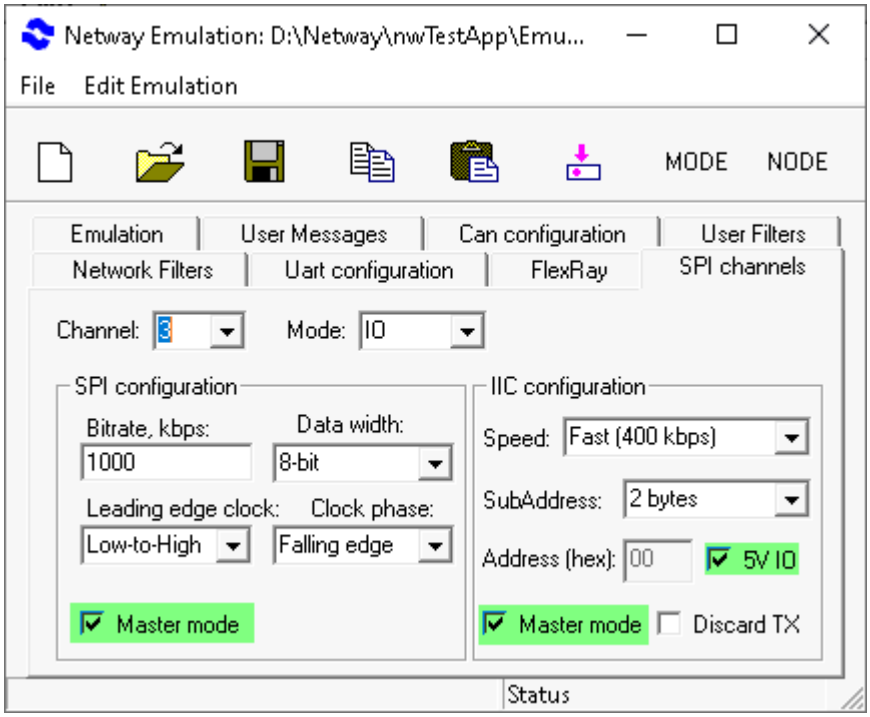
Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame	Description
2	1:09:26.643	canl	TxCanl	8	101 15 02 03 04 05 06 07 08	
3	1:09:26.645	canl	canlflt10	8	201 15 02 03 04 05 06 07 08	
4	1:09:26.698	fray	frObj5	8/14	06C/108 11 22 00 44 55 66 77 88	STEA_OFFS
5	1:09:26.709	fray	frObj4	8/16	075/117 09 01 00 44 55 66 77 88	ST_CENG
6	1:09:26.718	fray	frObj5	8/18	06C/108 11 22 01 44 55 66 77 88	STEA_OFFS
7	1:09:26.728	fray	frObj4	8/20	075/117 A8 02 01 44 55 66 77 88	ST_CENG
8	1:09:26.738	fray	frObj5	8/22	06C/108 11 22 02 44 55 66 77 88	STEA_OFFS
9	1:09:26.748	fray	frObj4	8/24	075/117 3F 03 02 44 55 66 77 88	ST_CENG
10	1:09:26.759	fray	frObj5	8/26	06C/108 11 22 03 44 55 66 77 88	STEA_OFFS
11	1:09:26.768	fray	frObj4	8/28	075/117 F7 04 03 44 55 66 77 88	ST_CENG
12	1:09:26.778	fray	frObj5	8/30	06C/108 11 22 04 44 55 66 77 88	STEA_OFFS
13	1:09:26.788	fray	frObj4	8/32	075/117 65 05 04 44 55 66 77 88	ST_CENG
14	1:09:26.798	fray	frObj5	8/34	06C/108 11 22 05 44 55 66 77 88	STEA_OFFS
15	1:09:26.809	fray	frObj4	8/36	075/117 C4 06 05 44 55 66 77 88	ST_CENG
16	1:09:26.818	fray	frObj5	8/38	06C/108 11 22 06 44 55 66 77 88	STEA_OFFS
17	1:09:26.828	fray	frObj4	8/40	075/117 53 07 06 44 55 66 77 88	ST_CENG
18	1:09:26.838	fray	frObj5	8/42	06C/108 11 22 07 44 55 66 77 88	STEA_OFFS
19	1:09:26.849	fray	frObj4	8/44	075/117 49 08 07 44 55 66 77 88	ST_CENG
20	1:09:26.858	fray	frObj5	8/46	06C/108 11 22 08 44 55 66 77 88	STEA_OFFS
21	1:09:26.868	fray	frObj4	8/48	075/117 D1 09 08 44 55 66 77 88	ST_CENG
22	1:09:26.878	fray	frObj5	8/50	06C/108 11 22 09 44 55 66 77 88	STEA_OFFS
23	1:09:26.888	fray	frObj4	8/52	075/117 70 0A 09 44 55 66 77 88	ST_CENG
24	1:09:26.898	fray	frObj5	8/54	06C/108 11 22 0A 44 55 66 77 88	STEA_OFFS
25	1:09:26.908	fray	frObj4	8/56	075/117 E7 0B 0A 44 55 66 77 88	ST_CENG
26	1:09:26.918	fray	frObj5	8/58	06C/108 11 22 0B 44 55 66 77 88	STEA_OFFS
27	1:09:26.928	fray	frObj4	8/60	075/117 2F 0C 0B 44 55 66 77 88	ST_CENG
28	1:09:26.938	fray	frObj5	8/62	06C/108 11 22 0C 44 55 66 77 88	STEA_OFFS
29	1:09:26.949	fray	frObj4	8/0	075/117 BD 0D 0C 44 55 66 77 88	ST_CENG
30	1:09:26.958	fray	frObj5	8/2	06C/108 11 22 0D 44 55 66 77 88	STEA_OFFS
31	1:09:26.968	fray	frObj4	8/4	075/117 1C 0E 0D 44 55 66 77 88	ST_CENG
32	1:09:26.978	fray	frObj5	8/6	06C/108 11 22 0E 44 55 66 77 88	STEA_OFFS
33	1:09:26.988	fray	frObj4	8/8	075/117 D7 00 0E 44 55 66 77 88	ST_CENG
34	1:09:26.999	fray	frObj5	8/10	06C/108 11 22 0F 44 55 66 77 88	STEA_OFFS
35	1:09:27.008	fray	frObj4	8/12	075/117 CC 01 0F 44 55 66 77 88	ST_CENG
36	1:09:27.049	fray	frObj4	8/20	075/117 8D 02 FF 08 09 0A 0B 0C	ST_CENG
37	1:09:27.050	canl	TxCanl	8	101 12 02 03 04 05 06 07 08	
38	1:09:27.051	canl	canlflt10	8	201 12 02 03 04 05 06 07 08	
39	1:09:27.152	spi3	flt08	8	20 21 22 23 24 25 26 00	
40	1:09:27.152	spi3	flt08	8	20 21 22 23 24 25 26 01	

Item 2 shows Box1 request - start of Flexray test
Item 3 shows Box2 response - ready for Flexray test
Items 4-35:
TX: 06C 11 22 CNT 44 55 66 77 88 - CNT = 00-:-0F
RX: 075 KA CRC CNT44 55 66 77 88 - KA: keep-alive, CRC, CNT = CNT from previous
Item 36: 075 KA CRC FF 08 09 0A 0B 0C - bytes from slot 55 end of Flexray test!

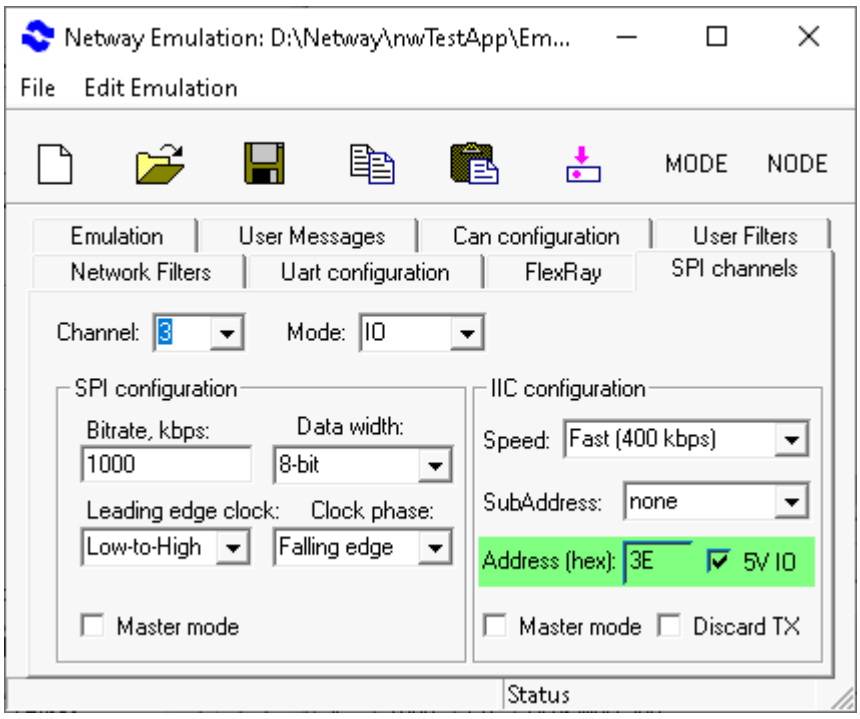
Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

SPI Test

Box 1 SPI configuration for channel 3 : SPI: Master mode, 5V IO; I2C: Master mode, 5V IO.



Box 2 SPI configuration for channel 3: SPI: Slave mode, 5V IO; I2C: Slave mode, 5V IO, Address: 3E


Netway Emulation: D:\Netway\nwTestApp\Em...
File
Edit Emulation
MODE
NODE
Emulation
User Messages
Can configuration
User Filters
Network Filters
Uart configuration
FlexRay
SPI channels
Channel: 3
Mode: IO
SPI configuration
Bitrate, kbps:
1000
Data width:
8-bit
Leading edge clock:
Low-to-High
Clock phase:
Falling edge
Master mode
IIC configuration
Speed: Fast (400 kbps)
SubAddress: none
Address (hex): 3E
5V IO
Master mode
Discard TX
Status

Step 26: SPI Test

can1flt10->211 filter for Box2 protocol ready response

Event81: Start Step26

```
ENABLE protocol318; //enable channel3 SPI protocol in this box SET
umsgbyte600 = 18; //verify SPI protocol in box2 by CAN message QUE
usermsg60; //TX CAN message
SET timer1 = 2000; //allow 2 seconds for test
```

Event82: can1flt10->

```
//Response from Box2 -> make sure Box2 is SPI ready:
IF rxdata0 == 15; //if I2C protocol
CL cl-tag93; //Box2 is ready start I2C test
ELSEIF rxdata0 == 18; //if SPI protocol
CL cl-tag97; //box2 is ready start SPI test
ENDIF;
```

Event88: Start SPI Test

```
SET x10 = 0; //reset count
SET delay4 = 100; //Delay 100 ms
SET umsgbyte697 = x10; //count to 7th byte of user message69
QUE usermsg69; //TX user message69
```

Event89: filter8-> SPI->20 21 22 23 -- -- -- -- -- -- --

```
IF x10 < 5; //if less than 5 messages has been received SET x10 + 1;
//increment count
SET umsgbyte697 = x10; //count to byte 7 of user message69
QUE usermsg69; //TX user message 69
ELSE; //all messages has been received
IF x10 == 5; //if count is 5
    IF rxdata7 == 5; //if rxdata7 = x10
    CL cl-tag75; //go to next test step 27
```

```
    ELSE; CL cl-tag9; //data miss-match ENDIF; ENDIF;
ENDIF;
```

Step 27: I2C Test

We use user message65 and user message66 to perform I2C test: 65: iic3-> 3E 11 12 13 14 15 16 17 XX - XX:count 1-:-5 Write to Slave 66: iic3-> 3E 01 02 3F Y1 Y2 Y3 Y4 Y5 - Read from Slave. Yn - slave data

Event86: Initialize I2C Test

```
SET timer1 = 2000; //allow 2 seconds for test
DISABLE protocol315; //disable I2C protocol
ENABLE protocol315; //enable I2C protocol in this box
SET umsgbyte600 = 15; //verify I2C protocol in box2 by CAN message
QUE usermsg60; //TX user message60
```

Event83: Start I2C Test

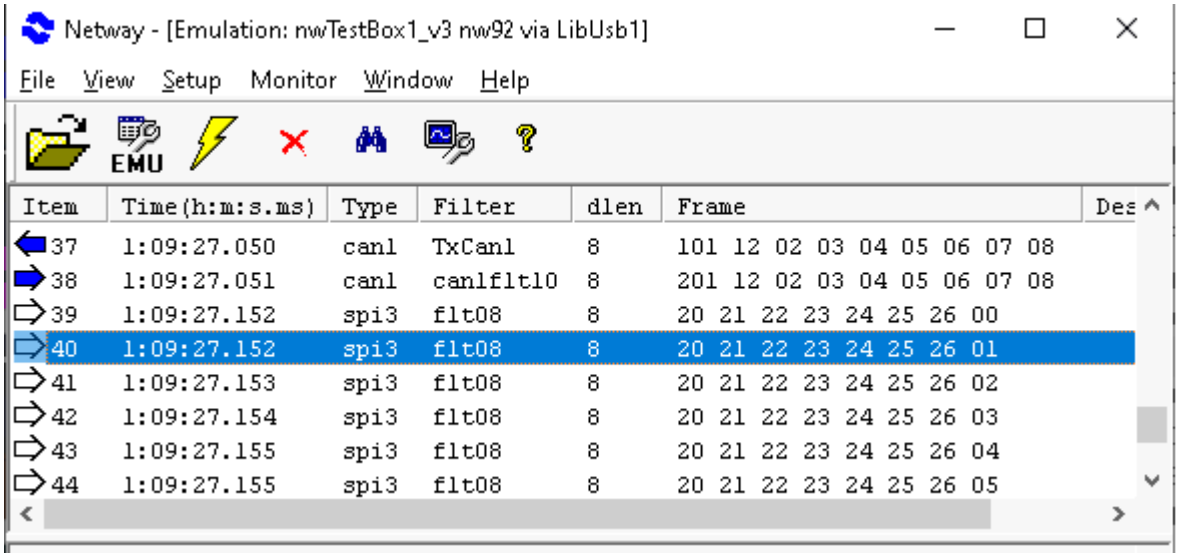
```
SET x10 = 0; //reset count
SET delay4 = 100; //Delay 100 ms
SET umsgbyte658 = x10; //count to 8th byte of user message65
QUE usermsg65; //TX user message65 - Write to Slave
SET delay4 = 200; //Delay for 200 ms
SET umsgbyte668 = x10; //count to 8th byte of user message66
QUE usermsg66; //TX user message66 - Read from Slave
```

Event84: filter8-> I2C->3E 01 02 3F 20 -- -- -- -- --

```
IF x10 < 4; //if less than 5 messages received
SET x10 + 1; //increment count
SET umsgbyte658 = x10; //count the 8th byte of user message65
QUE usermsg65; //TX user message65 - Write to Slave
SET delay4 = 150; //allow slave to set response
QUE usermsg66; //TX user message66 - Read from Slave
ELSE; //5 messages has been received
IF rxdata8 == 5; //if 8th byte of last RX message equals 5
    SET timer1 = 0; //reset time-out; CL cl-tag8; //end of test
ELSE;
    CL cl-tag9; //data mismatch
```

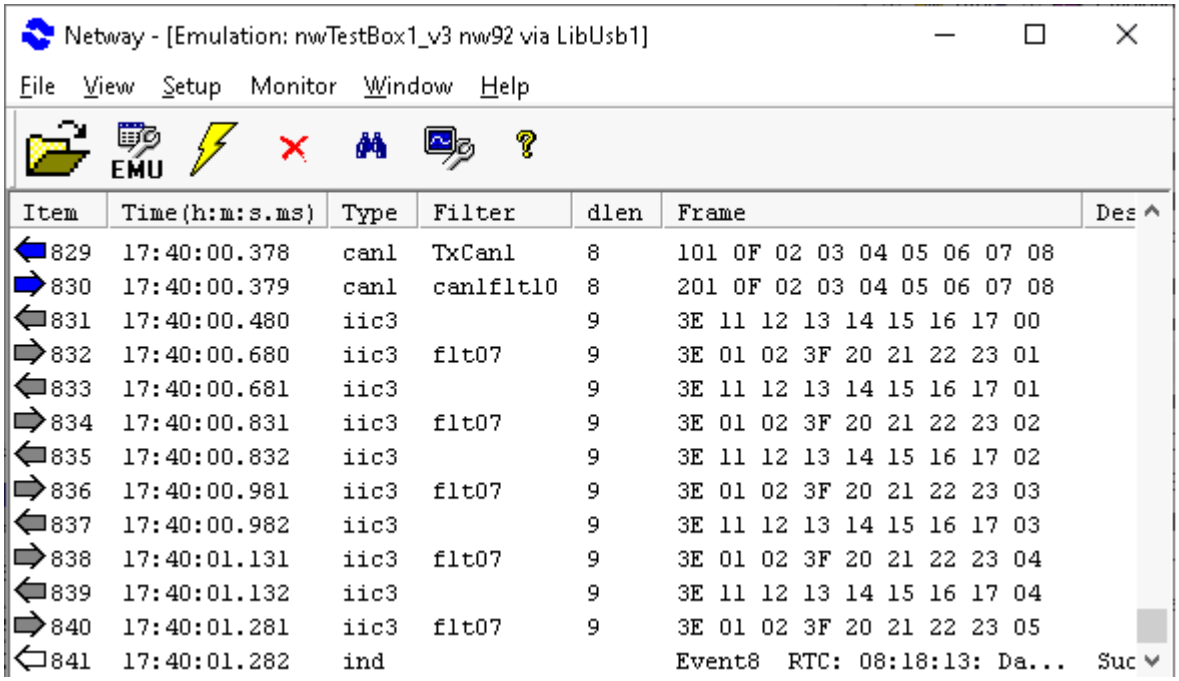
ENDIF;
ENDIF;

SPI Test Log:



Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame	Des
37	1:09:27.050	canl	TxCanl	8	101 12 02 03 04 05 06 07 08	
38	1:09:27.051	canl	canflt10	8	201 12 02 03 04 05 06 07 08	
39	1:09:27.152	spi3	flt08	8	20 21 22 23 24 25 26 00	
40	1:09:27.152	spi3	flt08	8	20 21 22 23 24 25 26 01	
41	1:09:27.153	spi3	flt08	8	20 21 22 23 24 25 26 02	
42	1:09:27.154	spi3	flt08	8	20 21 22 23 24 25 26 03	
43	1:09:27.155	spi3	flt08	8	20 21 22 23 24 25 26 04	
44	1:09:27.155	spi3	flt08	8	20 21 22 23 24 25 26 05	

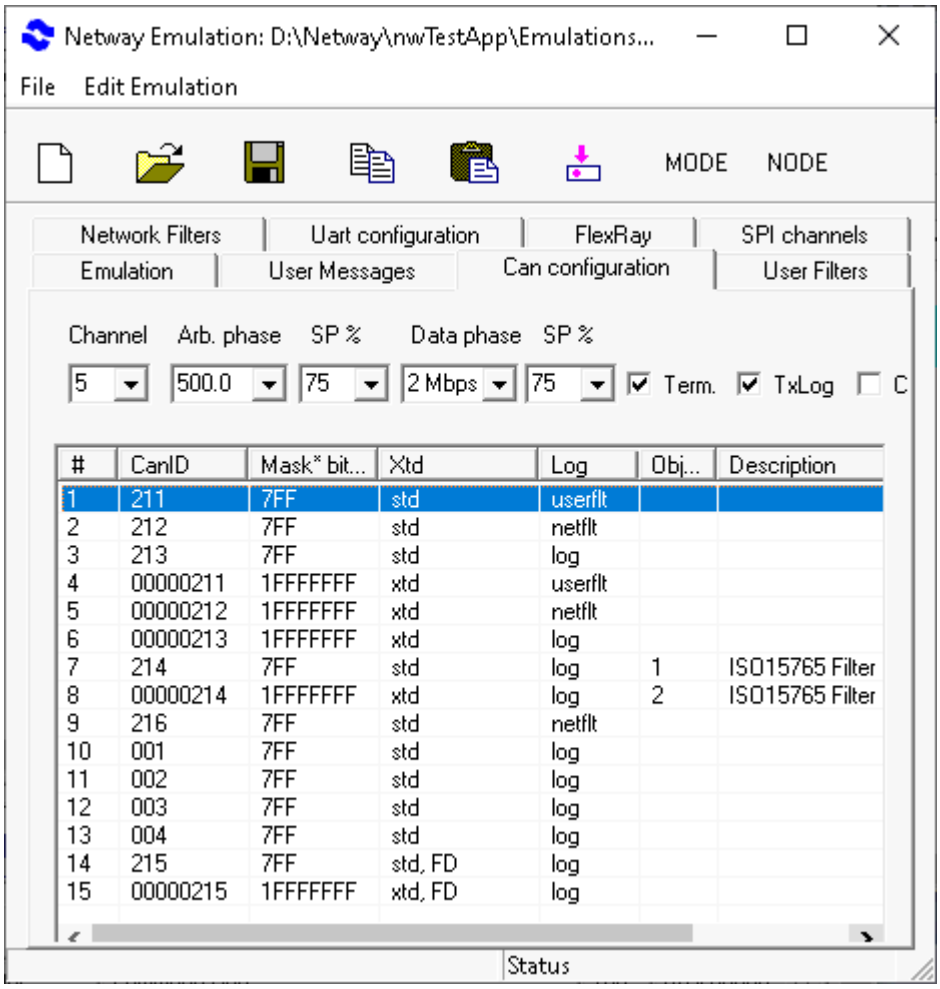
I2C Test Log:



Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame	Des
829	17:40:00.378	canl	TxCanl	8	101 0F 02 03 04 05 06 07 08	
830	17:40:00.379	canl	canflt10	8	201 0F 02 03 04 05 06 07 08	
831	17:40:00.480	iic3		9	3E 11 12 13 14 15 16 17 00	
832	17:40:00.680	iic3	flt07	9	3E 01 02 3F 20 21 22 23 01	
833	17:40:00.681	iic3		9	3E 11 12 13 14 15 16 17 01	
834	17:40:00.831	iic3	flt07	9	3E 01 02 3F 20 21 22 23 02	
835	17:40:00.832	iic3		9	3E 11 12 13 14 15 16 17 02	
836	17:40:00.981	iic3	flt07	9	3E 01 02 3F 20 21 22 23 03	
837	17:40:00.982	iic3		9	3E 11 12 13 14 15 16 17 03	
838	17:40:01.131	iic3	flt07	9	3E 01 02 3F 20 21 22 23 04	
839	17:40:01.132	iic3		9	3E 11 12 13 14 15 16 17 04	
840	17:40:01.281	iic3	flt07	9	3E 01 02 3F 20 21 22 23 05	
841	17:40:01.282	ind			Event8 RTC: 08:18:13: Da...	Suc

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

CAN5/6 Test



#	CanID	Mask* bit...	Xtd	Log	Obj...	Description
1	211	7FF	std	userflt		
2	212	7FF	std	netflt		
3	213	7FF	std	log		
4	00000211	1FFFFFFF	xtd	userflt		
5	00000212	1FFFFFFF	xtd	netflt		
6	00000213	1FFFFFFF	xtd	log		
7	214	7FF	std	log	1	ISO15765 Filter
8	00000214	1FFFFFFF	xtd	log	2	ISO15765 Filter
9	216	7FF	std	netflt		
10	001	7FF	std	log		
11	002	7FF	std	log		
12	003	7FF	std	log		
13	004	7FF	std	log		
14	215	7FF	std, FD	log		
15	00000215	1FFFFFFF	xtd, FD	log		

CAN configuration channel 5 is shown above.

Step 28: CAN5 Flter9 Test

Box1 user message71: can5->111 00 00 00 00 00 00 00 00
Box1 user filter9: can5->211 -- -- -- -- -- -- -- --
Box1 sends can5->111 NN 00 00 00 00 00 00 00
Box2 responds with can5->211 NN 00 00 00 00 00 00 00 (flt9) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event91: Start Step 28

```
IF protocol8 == 0h; //CAN5 disabled
CL cl-tag10; //Protocol is not available
ENDCL;
ENDIF;
LOGEVNT; //log event
SET x1 = 0h; //reset count
SET x9 = 0h; SET umsgbyte710 = x9; //count to first byte of user message71
QUE usermsg71; //TX user message 71
SET y3 = 0h; //reset messages masks
SET timer1 = 1000; //allow 1 second for test
```

Event92: filter9->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for this RX message
IF x1 == x9; //if match to counter
IF x9 < Fh; //less than 16 messages has been received SET x9 + 1h; //increment counter SET umsgbyte710 = x9; //counter to first byte of user message 71
    QUE usermsg71; //TX user message 71
ELSE; //counter = 16
    IF y3 == FFFFh; //all 16 messages has been received CL offset+1; //go to next step test ELSE; CL cl-tag2; //missed message ENDIF; ENDIF;
ENDIF;
```

Step 29: CAN5 netflt11 Test

Box1 user message72: can5->112 00 00 00 00 00 00 00 00
Box1 user filter1: can5->212 -- -- -- -- -- -- -- --
Box1 sends can5->112 NN 00 00 00 00 00 00 00
Box2 responds with can5->212 NN 00 00 00 00 00 00 00 (netflt11) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event93: Start Step 29

```
IF protocol8 == 0h; //CAN5 disabled
CL cl-tag10; //Protocol is not available
ENDCL;
ENDIF;
LOGEVNT; //log event
SET x1 = 0h; //reset count
SET x9 = 0h; SET umsgbyte720 = x9; //count to first byte of user message72
QUE usermsg72; //TX user message 72
SET y3 = 0h; //reset messages masks
SET timer1 = 1000; //allow 1 second for test
```

Event94: netflt11->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for this RX message
IF x1 == x9; //if match to counter
IF x9 < Fh; //less than 16 messages has been received SET x9 + 1h; //increment counter SET umsgbyte720 = x9; //counter to first byte of user message 72
    QUE usermsg72; //TX user message 72
ELSE; //counter = 16
    IF y3 == FFFFh; //all 16 messages has been received CL offset+1; //go to next step test ELSE; CL cl-tag2; //missed message ENDIF; ENDIF;
ENDIF;
```

Step 30: CAN5 can5lft3 Test

Box1 user message73: can5->113 00 00 00 00 00 00 00 00
Box1 can filter1: can5->213-- -- -- -- -- -- -- --
Box1 sends can5->113 NN 00 00 00 00 00 00 00
Box2 responds with can5->213 NN 00 00 00 00 00 00 00 (can5flt3) NN = 0 -:- 0F. 16 messages transmitted, 16 messages received

Event95: Start Step 29

```
LOGEVNT; //log event
SET x9 = 0h; //reset counter
SET umsgbyte730 = x9; //count to first byte of user message73
QUE usermsg73; //TX user message 73
SET y3 = 0h; //reset messages masks
SET timer1 = 100; //allow 100 ms for test
```

Event96: can5flt3->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for this RX message
IF x1 == x9; //if match to counter
IF x9 < Fh; //less than 16 messages has been received SET x9 + 1h; //increment counter SET umsgbyte730 = x9; //counter to first byte of user message 73
    QUE usermsg73; //TX user message 73
ELSE; //counter = 16
    IF y3 == FFFFh; //all 16 messages has been received CL offset+1; //go to next step test ELSE; CL cl-tag2; //missed message ENDIF; ENDIF;
ENDIF;
```

Step 31: CAN5 Xtd filter10 Test

Box1 user message74: can5->0000111 00 00 00 00 00 00 00 00
Box1 user filter10: can5->0000211 -- -- -- -- -- -- -- --
Box1 sends can5->0000111 NN 00 00 00 00 00 00 00
Box2 responds with can5->0000211 NN 00 00 00 00 00 00 00 (fiter10) NN = 0 -:- 0F. 16 messages transmitted

Event97: Start Step 31

```
IF protocol8 == 0h; //CAN5 disabled
CL cl-tag10; //Protocol is not available
ENDCL;
ENDIF;
LOGEVNT; //log event
SET x1 = 0h; //reset count
SET x9 = 0h; SET y3 = 0h; //reset messages masks
SET umsgbyte740 = x9; //count to first byte of user message74
QUE usermsg74; //TX user message 74
SET timer1 = 100;
```

Event98: filter10->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for this RX message
IF x1 == x9; //if match to counter
IF x9 < Fh; //less than 16 messages has been received SET x9 + 1h; //increment counter SET umsgbyte740 = x9; //counter to first byte of user message 74
    QUE usermsg74; //TX user message 74
ELSE; //counter = 16
    IF y3 == FFFFh; //all 16 messages has been received CL offset+1; //go to next step test ELSE; CL cl-tag2; //missed message ENDIF; ENDIF;
ENDIF;
```

Step 32: CAN5 Xtd netflt12 Test

Box1 user message75: can5->0000112 00 00 00 00 00 00 00 00
Box1 network filter12: can5->0000212 -- -- -- -- -- -- -- --
Box1 sends can5->0000112 NN 00 00 00 00 00 00 00
Box2 responds with can5->0000212 NN 00 00 00 00 00 00 00 (netflt12)
NN = 0 -:- 0F. 16 messages transmitted

Event99: Start Step 32

```
IF protocol8 == 0h; //CAN5 disabled
CL cl-tag10; //Protocol is not available
ENDCL;
ENDIF;
LOGEVNT; //log event
SET x1 = 0h; //reset count
SET x9 = 0h; SET y3 = 0h; //reset messages masks
SET umsgbyte750 = x9; //count to first byte of user message75
QUE usermsg75; //TX user message 75
SET timer1 = 100;
```

Event100: netflt12->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for this
RX message
IF x1 == x9; //if match to counter
IF x9 < Fh; //less than 16 messages has been received SET x9 + 1h;
//increment counter SET umsgbyte750 = x9; //counter to first byte of user
message 75
    QUE usermsg75; //TX user message 75
ELSE; //counter = 16
    IF y3 == FFFFh; //all 16 messages has been received CL offset+1; //go
 to next step test ELSE; CL cl-tag2; //missed message ENDIF; ENDIF;
ENDIF;
```

Step 33: CAN5 Xtd can5flt6 Test

Box1 user message76: can5->0000113 00 00 00 00 00 00 00 00
Box1 network filter12: can5->0000213 -- -- -- -- -- -- -- --
Box1 sends can5->0000113 NN 00 00 00 00 00 00 00
Box2 responds with can5->0000213 NN 00 00 00 00 00 00 00 (can5flt6)
NN = 0 -:- 0F. 16 messages transmitted

Event99: Start Step 33

```
SET x9 = 0h; //reset count
SET y3 = 0h; //reset messages masks
SET umsgbyte760 = x9; //count to first byte of user message76
QUE usermsg76; //TX user message 76
SET timer1 = 100;
```

Event102: can5flt6->

```
SET x1 = rxdata0; //first RX byte to x1
SET y1 = x1; SET y2 = 1h; SET y2 << y1; SET y3 | y2; //set mask for this
RX message
IF x1 == x9; //if match to counter
IF x9 < Fh; //less than 16 messages has been received SET x9 + 1h;
//increment counter SET umsgbyte760 = x9; //counter to first byte of user
message 76
    QUE usermsg76; //TX user message 76
ELSE; //counter = 16
    IF y3 == FFFFh; //all 16 messages has been received CL offset+1; //go
 to next step test ELSE; CL cl-tag2; //missed message ENDIF; ENDIF;
ENDIF;
```

Step 34: CAN5 Std Multi-Frame Test

Box1: fill out MF1 64-data bytes from umsgs 8,9,10: can1->114 10 11...4B 4C 4D 4E 4F
and TX MF1 to Box2.
Box2 detects MF1 and send back 64-byte message back: can1->214 -- -- -- -- -- -

Event103: Start Step 34

```
LOGEVNT; // log event
SET umsgbyte80 = 10h; //restore original first user message8 byte SET timer1 = 100; //allow 100 ms for test
SET TxLong1 = bytesize64; //64-bytes for TX long message SET TxLong1 = usermsg8; //copy data bytes from user messages 8,9,10
SET mfCan105 = TxLong1; //assign TX long to MF1 CAN5 sub-channel 5
SET mfCan105 = bytesize64; //Set MF1 CAN5 to 64 bytes QUE mfCan105; //TX MF1 CAN5 sub-channel 5
```

Event25: mfCanRx105 ->

```
IF mfCan5 == usermsg8; //if received data equals to content of usermsg8,9,10
CL offset+1; //go to next step
ELSE; //not equals
CL cl-tag4; //test failed
ENDIF;
```

Step 35: CAN5 Xtd Multi-Frame Test

Box1: fill out MF1 64-data bytes from umsgs 8,9,10: can1->0000114 10 11...4B 4C 4D 4E 4F
and TX MF2 to Box2.
Box2 detects MF2 and send back 64-byte message back: can1->0000214 -- -- -- -- -- -

Event105: Start Step 35

```
LOGEVNT; // log event
SET umsgbyte80 = 10h; //restore original first user message8 byte SET timer1 = 100; //allow 100 ms for test
SET TxLong1 = bytesize64; //64-bytes for TX long message SET TxLong1 = usermsg8; //copy data bytes from user messages 8,9,10
SET mfCan205 = TxLong1; //assign TX long to MF2 CAN5 sub-channel 5
SET mfCan205 = bytesize64; //Set MF2 CAN5 to 64 bytes QUE mfCan205; //TX MF2 CAN5 sub-channel 5
```

Event106: mfCanRx205 ->

```
IF mfCan5 == usermsg8; //if received data equals to content of usermsg8,9,10
CL offset+1; //go to next step
ELSE; //not equals
CL cl-tag4; //test failed
ENDIF;
```

Step 36: Std CAN5 FD Test

Box1: TX usermsg77 (CAN FD) can5->115 00 01 ...3E 3F (64 bytes) Box2: Responds with (CAN FD) can5->215 00 01...3E 3F (64 bytes) can5flt14 Box1 verifies selected RX bytes and if match test pass

Event107: Start Step 36

```
SET timer1 = 100; //allow 100 ms for test
IF deviceState18 > 1Dh; //if hardware is NW92 and above LOGEVNT;
//log event
QUE usermsg77; //TX CAN FD message
ELSE; //if not CAN FD hardware
//CL cl-tag32; //skip CAN FD test, go to step 11
ENDIF;
```

Event108: can5flt14 ->

```
IF rxdata15 != Fh; //verify RX byte 16
SET x1 = Fh; SET x2 = rxdata15; CL cl-tag5; //byte 16 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata39 != 27h; //verify RX byte 40
SET x1 = 27h; SET x2 = rxdata39; CL cl-tag5; //byte 40 not valid, test
failed
ENDCL; //stop verification
ENDIF;
IF rxdata63 != 3Fh; //verify RX byte 64
SET x1 = 3Fh; SET x2 = rxdata63; CL cl-tag5; //byte 64 not valid, test
failed
ENDCL; //stop verification
ENDIF;
CL offset+1; //Ok, go to step 10
```

Step 37: Std CAN5 Xtd FD Test

Box1: TX usermsg81 (CAN FD) can5->0000115 00 01 ...3E 3F (64 bytes)
Box2: Responds with (CAN FD) can5->0000215 00 01...3E 3F (64 bytes) can5flt15
Box1 verifies selected RX bytes and if match test pass

Event109: Start Step 37

```
SET timer1 = 100; //allow 100 ms for test
IF deviceState18 > 1Dh; //if hardware is NW92 and above LOGEVNT; // log event
QUE usermsg81; //TX CAN FD message
ELSE; //if not CAN FD hardware
//CL cl-tag32; //skip CAN FD test, go to step 11
ENDIF;
```

Event110: can5flt14 ->

```
IF rxdata15 != Fh; //verify RX byte 16
SET x1 = Fh; SET x2 = rxdata15; CL cl-tag5; //byte 16 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata39 != 27h; //verify RX byte 40
SET x1 = 27h; SET x2 = rxdata39; CL cl-tag5; //byte 40 not valid, test failed
ENDCL; //stop verification
ENDIF;
IF rxdata63 != 3Fh; //verify RX byte 64
SET x1 = 3Fh; SET x2 = rxdata63; CL cl-tag5; //byte 64 not valid, test failed
ENDCL; //stop verification
ENDIF;
CL offset+1; //Ok, go to step 10
```

CAN5 Test Log:

Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame	De
1	15:56:31.114	ind			Event91 RTC: 05:52:33: Date: 03/...	S1
2	15:56:31.116	can5	TxCan5	8	111 00 00 00 00 00 00 00 00	
3	15:56:31.118	can5	flt09	8	211 00 00 00 00 00 00 00 00	
4	15:56:31.120	can5	TxCan5	8	111 01 00 00 00 00 00 00 00	
5	15:56:31.122	can5	flt09	8	211 01 00 00 00 00 00 00 00	
6	15:56:31.124	can5	TxCan5	8	111 02 00 00 00 00 00 00 00	
7	15:56:31.126	can5	flt09	8	211 02 00 00 00 00 00 00 00	
8	15:56:31.128	can5	TxCan5	8	111 03 00 00 00 00 00 00 00	
9	15:56:31.130	can5	flt09	8	211 03 00 00 00 00 00 00 00	
10	15:56:31.132	can5	TxCan5	8	111 04 00 00 00 00 00 00 00	
11	15:56:31.134	can5	flt09	8	211 04 00 00 00 00 00 00 00	
12	15:56:31.136	can5	TxCan5	8	111 05 00 00 00 00 00 00 00	
13	15:56:31.138	can5	flt09	8	211 05 00 00 00 00 00 00 00	
14	15:56:31.141	can5	TxCan5	8	111 06 00 00 00 00 00 00 00	
15	15:56:31.142	can5	flt09	8	211 06 00 00 00 00 00 00 00	
16	15:56:31.145	can5	TxCan5	8	111 07 00 00 00 00 00 00 00	
17	15:56:31.146	can5	flt09	8	211 07 00 00 00 00 00 00 00	
18	15:56:31.148	can5	TxCan5	8	111 08 00 00 00 00 00 00 00	
19	15:56:31.151	can5	flt09	8	211 08 00 00 00 00 00 00 00	
20	15:56:31.153	can5	TxCan5	8	111 09 00 00 00 00 00 00 00	

Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame
236	15:56:31.575	can5	can5flt08	8	00000214 22 1D 1E 1F 20 21 22 C7
237	15:56:31.577	can5	can5flt08	8	00000214 23 24 25 26 27 28 29 C7
238	15:56:31.577	can5	can5flt08	8	00000214 24 2B 2C 2D 2E 2F 30 C7
239	15:56:31.578	can5	can5flt08	8	00000214 25 32 33 34 35 36 37 C7
240	15:56:31.580	can5	can5flt08	8	00000214 26 39 3A 3B 3C 3D 3E C7
241	15:56:31.580	can5	can5flt08	8	00000214 27 40 41 42 43 44 45 C7
242	15:56:31.581	can5	can5flt08	8	00000214 28 47 48 49 4A 4B 4C C7
243	15:56:31.583	can5	can5flt08	3	00000214 29 4E 4F
244	15:56:31.583	ind			Event107 RTC: 05:52:33: Date: 03...
245	15:56:31.586	can5	TxCan5	16	115 00 01 02 03 04 05 06 07 08 09...
246				24	10 11 12 13 14 15 16 17 18 19 1A ...
247				24	28 29 2A 2B 2C 2D 2E 2F 30 31 32 ...
248	15:56:31.589	can5	can5flt14	16	215 00 01 02 03 04 05 06 07 08 09...
249				24	10 11 12 13 14 15 16 17 18 19 1A ...
250				24	28 29 2A 2B 2C 2D 2E 2F 30 31 32 ...
251	15:56:31.593	can5	TxCan5	16	00000115 00 01 02 03 04 05 06 07 ...
252				24	10 11 12 13 14 15 16 17 18 19 1A ...
253				24	28 29 2A 2B 2C 2D 2E 2F 30 31 32 ...
254	15:56:31.596	can5	can5flt15	16	00000215 00 01 02 03 04 05 06 CE ...
255				24	10 11 12 13 14 15 16 17 18 19 1A ...
256				24	28 29 2A 2B 2C 2D 2E 2F 30 31 32 ...

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

J1708/J1939 Test

Box1: Change Uart1 Raw protocol to J1708
Set PID 84, 190 (J1708)
Set SPN 84,190 (J1939)
TX user message 60: can1->101 F1 ... to signal protocol switch in Box2

Box2: TX user messages 95,96,97:
95: ubp1-> 80 5C 1E BE C0 12 54 1E 5B 1E (PIDs 84, 190) 96: can1-> 0CF00400 FF FF FF 41 1F FF FF FF(SPN 190)
97: can2-> 18FEF100 FF 60 40 FC FF FF FF FF(SPN 84)

Step 47: Start J1708/J1939 Test

```
LOGEVNT;
SET uartType1 = 7; //J1708
SET umsgbyte600 = F1h; //J1708 / J1939 test
QUE usermsg60;
//J1708 parameters
SET JPid84 = JMid128; //MID=80h=128, 1 byte - Road Speed 0.805
SET JPid84 = x20; //PID destination to x20
SET JPid190 = JMid128; //MID=80h=128, 2 bytes - RPM 0.25
SET JPid190 = x21; //PID destination to x21,22
//J1939 parameters
SET JSpn84 = JPgn65265; //PGN65265, 2 bytes - Vehicle speed 0.003906
SET JSpn84 = x23; //SPN destination to x23,24
SET JSpn190 = JPgn61444; //PGN61444, 2 bytes - RPM 0.125
SET JSpn190 = x25; //SPN destination to x251,26
SET timer1 = 1000; //allow 1 second for test
```

Event130: Verify J1708/J1939 Results: jpid84->

```
SET uartType1 = 6; //back to RAW protocol for Box1
SET umsgbyte600 = F2h; //back to RAW for Box2
QUE usermsg60; //TX user message 60 to Box2
IF x20 = 1Eh; //Verify received PIDs, SPNs
IF x21 = C0h;
```

```
    IF x22 = 12h;
    IF x23 = 60h;
    IF x24 = 40h;
    IF x25 = 41h;
    IF x26 = 1Fh;
    //match all parameters, end test
    SET uartType1 = 6; //RAW protocol
    SET umsgbyte600 = F2h; //back to RAW
    QUE usermsg60; //TX user message60
    ENDCL;
    ENDIF;
    ENDIF;
    ENDIF;
    ENDIF;
    ENDIF;
ENDIF;
ENDIF;
CL cl-tag9;
```

J1708 / J1939 Test Log:

Item	Time(h:m:s.ms)	Type	Filter	dlen	Frame	Description
1	15:50:08.134	ind			Event129 RTC: 05:46:10: Date: 03...	Step 47 - J1708-PID, J1939- PGN
2	15:50:08.135	canl	TxCanl	8	101 F1 02 03 04 05 06 07 08	
3	15:50:08.136	canl	canlfltl0	8	201 F1 02 03 04 05 06 07 08	
4	15:50:08.335	canl	canlfltl6	8	0CF00400 FF FF FF 41 1F FF FF CF	
5	15:50:08.335	canl	canlfltl6	8	18FEF100 FF 60 40 FC FF FF FF CF	
6	15:50:08.349	j1708_1		11	80 5C 1E BE C0 12 54 1E 5B 1E <8B>	
7	15:50:08.350	canl	TxCanl	8	101 F2 02 03 04 05 06 07 08	
8	15:50:08.351	canl	canlfltl0	8	201 F2 02 03 04 05 06 07 08	
9	15:50:08.351	ind			Event8 RTC: 05:46:10: Date: 03/2...	Successfully Finished Test

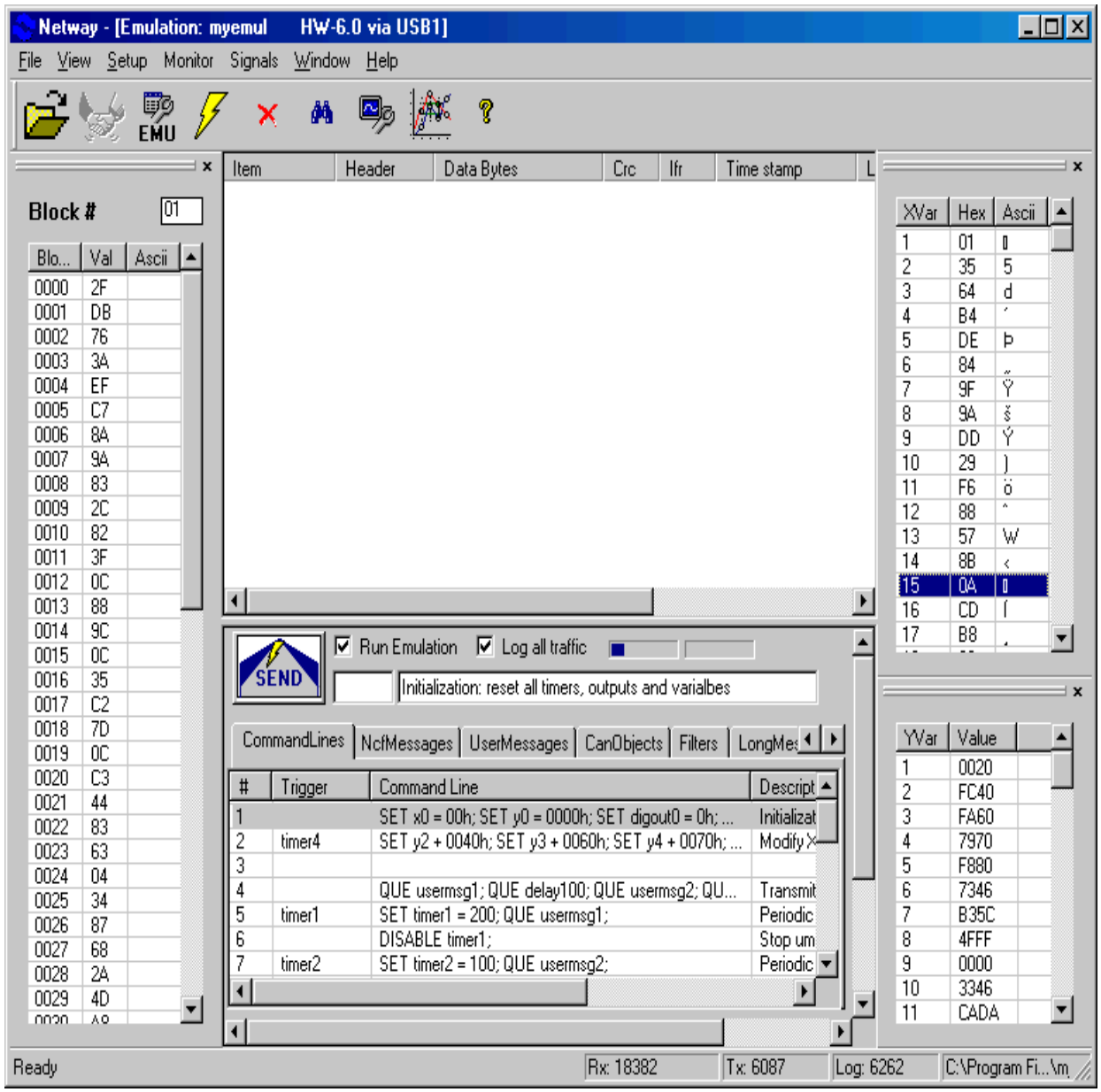
Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

Types

The *Netway Tool* includes several types of data storage for storing and manipulating data with the Netway Emulation Programming Language:

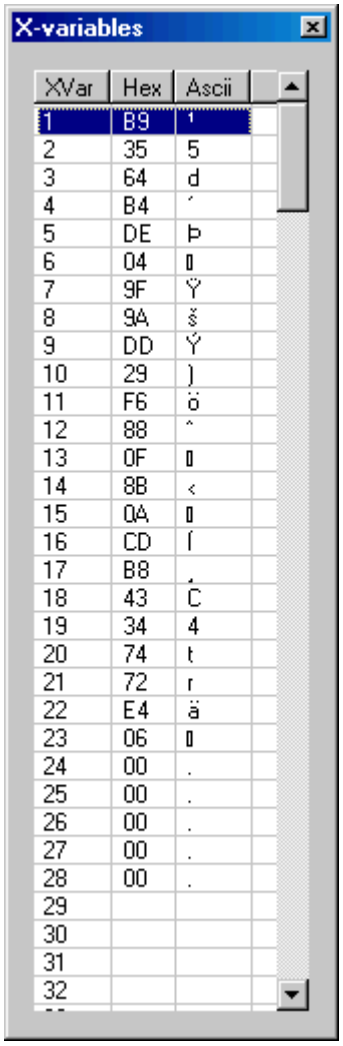
- X-Variables - 8-bit unsigned/signed values
- Y-Variables - 16-bit unsigned/signed values
- Z and NVZ-Variables - 32-bit unsigned/signed/float values
- Block Memory - 8-bit unsigned values

X,Y,Z Variables can be referenced in emulation program as signed or unsigned values.
xN, yN or zN are for referencing those variables as unsigned values, note N - variable number.
sxN, syN or szN are for referencing those variables as signed values, note N - variable number.
In addition Z Variables can be referenced as float values: fzN.
All the above items can be used to store independently created data, store data from received messages, and control program flow decisions. Furthermore, arithmetic and logical operations can be preformed on the data contained in all the above areas, and sets of neighboring storage cells from these areas can be copied into user message to be transmitted on any of the networks connected to a Netway device. Users can vies the X,Y,Z-Variable and Block Memory Windows as pictured below:


Netway - [Emulation: myemul HW-6.0 via USB1]
File View Setup Monitor Signals Window Help
EMU
Block # 01
Blo... Val Ascii
0000 2F
0001 DB
0002 76
0003 3A
0004 EF
0005 C7
0006 8A
0007 9A
0008 83
0009 2C
0010 82
0011 3F
0012 0C
0013 88
0014 9C
0015 0C
0016 35
0017 C2
0018 7D
0019 0C
0020 C3
0021 44
0022 83
0023 63
0024 04
0025 34
0026 87
0027 68
0028 2A
0029 4D
Item Header Data Bytes Crc Ifr Time stamp L
XVar Hex Ascii
1 01
2 35 5
3 64 d
4 B4 ´
5 DE Þ
6 84 „
7 9F Ÿ
8 9A š
9 DD Ý
10 29)
11 F6 ö
12 88 ˆ
13 57 W
14 8B ‹
15 0A
16 CD Í
17 B8 ¸
SEND
Run Emulation Log all traffic
Initialization: reset all timers, outputs and varialbes
CommandLines NcfMessages UserMessages CanObjects Filters LongMes
Trigger Command Line Descript
1 SET x0 = 00h; SET y0 = 0000h; SET digout0 = 0h; ... Initializat
2 timer4 SET y2 + 0040h; SET y3 + 0060h; SET y4 + 0070h; ... Modify X
3
4 QUE usermsg1; QUE delay100; QUE usermsg2; QU... Transmit
5 timer1 SET timer1 = 200; QUE usermsg1; Periodic
6 DISABLE timer1; Stop um
7 timer2 SET timer2 = 100; QUE usermsg2; Periodic
YVar Value
1 0020
2 FC40
3 FA60
4 7970
5 F880
6 7346
7 B35C
8 4FFF
9 0000
10 3346
11 CADA
Ready
Rx: 18382
Tx: 6087
Log: 6262
C:\Program Fi...\m

X-Variables

X-Variables are 8-bit data, and can therefore contain values ranging from $0-$FF (or 0-255 decimal).
The *Netway Tool* has 1024 *X-Variables* available for data storage and arithmetic manipulations.
X-Variables are referenced by the Emulation Programming Language as xN (unsigned) or sxN (signed), where N is from 0-256.
A special reference to the *X-Variables*, x0, is included in the Emulation Programming Language for the purposes of initializing all 256 *X-Variables* to one user preferred value. For example: the Command Statement SET x0 = 0; will set all 256 *X-Variables* to zero, while SET x0 = F0h; will set all variables to hex F0 value.
Example of signed operations: SET x1 = FFh; //-1
SET x2 = FEh; //-2
SET sx1 + sx2; //x1 = -3 or FD hex
Individual *X-Variables* can be set either with Command Statements (flow of emulation program) or with the *X-Variables* Window pictured below.



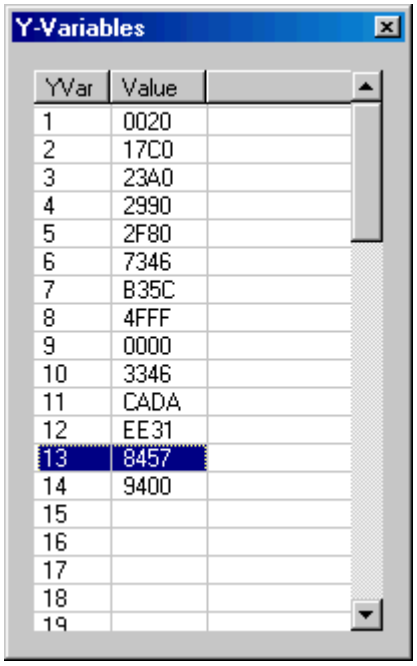
The *X-Variables* Window is accessed by selecting <View/X-Variables> from the menu bar of the *Netway Application Program*. Although the *X-Variables* Window will initially open as a separate window, it is dockable and can be incorporated into any side of the *Netway Application Program* main window simply by clicking on the title-bar of the *X-Variables* Window and dragging it to the desired location. The columns of the *X-Variables* Window display the index number of the variable, the hexadecimal value in the variable, and the ASCII character associated with that value. Users can manually enter the value of any of the 256 *X-Variables* simply by double-clicking in the desired Hex Column field, entering the desired value with the keyboard, and then pressing the Enter Key or clicking outside the active field.

Note: The *X-Variables* Window can only display 28 *X-Variables* at a time (notice that entries 29 and higher are blank in the picture above). Because

of this, users will not be able to manually enter the values of *X-Variables* at the bottom of the list if more than 28 *X-Variables* are displayed in the *X-Variables* Window. For example: in the window pictured above, one could not enter a value for x29 unless one either scrolled down or re-sized the window. To avoid this problem it is recommended that users resize the *X-Variables* Window to display a maximum of 28 *X-Variables*.

Y-Variables

Y-Variables are 2-byte (16-bit) data registers. The *Netway Tool* has 64 Y-Variables available for data storage and manipulation. *Y-Variables* are referenced by the Emulation Programming Language as yN, where N is from 1-64. A special reference to the *Y-Variables*, y0, is included in the Emulation Programming Language for the purposes of initializing all 64 *Y-Variables* to one value with a minimum amount of Command Statements. For example: the Command Statement SET y0 = 0000h; will set all 64 *Y-Variables* to zero.
Individual *Y-Variables* can be set either with Command Statements (see the SET Command section for more information) or with the *Y-Variables* Window pictured below.



The *Y-Variables* Window is accessed by selecting <View/Y-Variables> from the menu bar of the *Netway Application Program*. Although the *Y-Variables* Window will initially open as a separate window, it is dock-able and can be incorporated into the any side of the *Netway Application Program* main window simply by clicking on the title-bar of the *Y-Variables* Window and dragging it to the desired location. The columns of the *Y-Variables* Window

display the index number of the variable and the hexadecimal value in the variable. Users can manually enter the value of any of the 128 *Y-Variables* simply by double-clicking in the desired value column field, entering the desired value with the keyboard, and then pressing the Enter Key or clicking outside the active field.

Note: The *Y-Variables* Window can only display 14 *Y-Variables* at a time (notice that entries 15 and higher are blank in the picture above). Because of this, users will not be able to manually enter the values of *Y-Variables* at the bottom of the list if more than 14 *Y-Variables* are displayed in the *Y-Variables* Window. For example: in the window pictured above, one could not enter a value for y15 unless one either scrolled down or resized the window. To avoid this problem it is recommended that users resize the *Y-Variables* Window to display a maximum of 14 *Y-Variables*.

Z and NVZ-Variables

Z-Variables are 4-byte (32-bit) data registers. The *Netway Tool* has 64 *Z-Variables* available for data storage and manipulation. *Z-Variables* are referenced by the Emulation Programming Language as zN, where N is from 1-32. *The Z-Variables* could be set individually with Command Statements (see the SET Command section for more information).
NVZ-Variables are 4-byte (32-bit) data registers allocated in EEPROM memroy, this means they are non-volatile (power independent). The *Netway Tool* has 32 NV*Z-Variables* available for data storage and manipulation. *NVZ-Variables* are referenced by the Emulation Programming Language as nvzN, where N is from 1-32. *The NVZ-Variables* could be set individually with Command Statements (see the SET Command section for more information).
Note: The *Z-Variables* are primarily intended for internal system calculations, and cannot be viewed as X or Y-variables

Command Set

A Command Statement is expressed in one of the following forms:

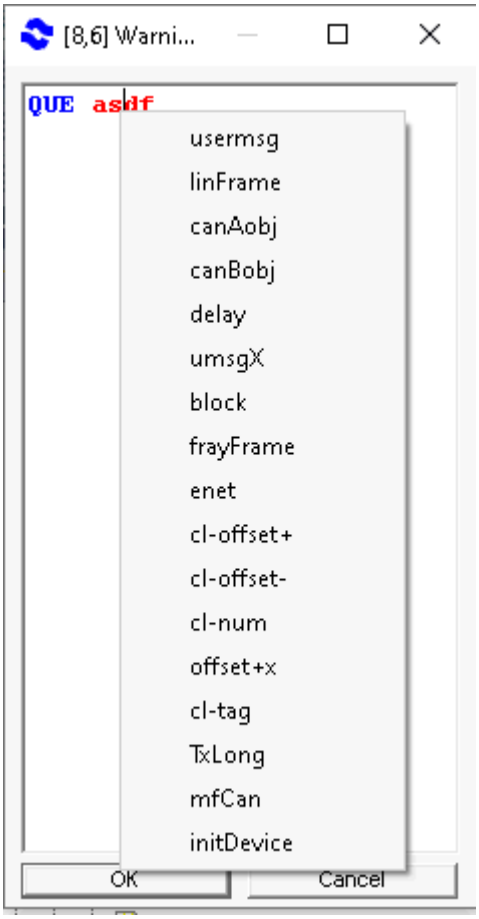
- No operand: **`Command;`**
- One Operand: **`Command TargetOperand;`**
- Two Operands: **`Command TargetOperand`** Operator `SourceOperand;`
- All command statements must end with semi column sign **:**

The following list contains all the Commands available in the Emulation Programming Language:

- **SET** – Two operand command used to define constants, copy data from received messages into memory, copy data from memory into transmittable messages, move data from one memory location to another, and perform arithmetic and logical operation on data.
- **QUE** – One operand command used to send a message into the transmit queue and partially control transmission timing.
- **LOOP** – One operand command, the operand defines number of loops to perform
- **ENABLE** – One operand command used primarily to enable Message Filters, Timers, logging, and Replay Mode.
- **DISABLE** – One operand command used primarily to disable Message Filters, Timers, logging, and Replay Mode.
- **IF** – Two operand command used to implement conditional programming statements and control program flow.
- **ELSE** – No operand command used to implement conditional programming statements and control program flow.
- **ELSEIF** – Two operand command used to implement conditional programming statements and control program flow.
- **ENDIF** – No operand command used to implement conditional programming statements and control program flow.
- **ENDCL** – No operand command used to terminate the execution of a Command Line.
- **CL** – One operand command used in a Command Line to execute a different Command Line as a subroutine.
- **NETMODE** – One operand command used to control various protocol specific features.

- **LOGEVNT** – No operand command used to send a time-stamped record of the current event to a Netway Device's internal log.
- **LOGANLG** – No operand command used to send a time-stamped record of the analog values of all the Digital/Analog Inputs to a Netway Device's internal log.
- **RESTTMR** – No operand command used to reset the Log Timer to zero.

While editing commands/operands typing any letters will result in red until match is found. Double-clicking on red will display possible matches as shown below:



SET Command

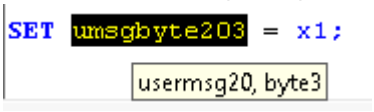
The general format of the **SET** command:

```
SET TargetOperand Operator SourceOperand;
```

The following examples illustrate variables/messages manipulations using **SET** command:

```
SET x7 = FFh;//assign fixed value x7=FFh
SET x7 >> 4; //shift x7 right four bits x7=0Fh
SET x7 + x2; //x7 = x7 + x2

SET y6 & AAh; //y6 = y6 AND AAh
SET y6 | y9; //y6 = y6 OR y9
SET y6 ~ y6; //y6 NOT y6

SET z1 = 1234h;//assign fixed value z1=1234h
SET z1 << 16;//shift z1 left 16 bits z1 = 12340000h
SET z1 | 5678h;//OR z1 with fixed value z1=12345678h

SET urmsgbyte203 = x1;//user message 20, byte 3 = x1

ENABLE bytesize4;//copy 4 bytes in next command
SET umsgbyte203 = x1;;//user message 20, bytes 3,4,5,6 =
x1,x2,x3,x4
```

When user highlights some operands, hint is displayed:




```
ENABLE bytesize8;//copy 8 bytes in next command SET x1 =
rxdata0;//copy 8 data bytes from RX message to x1-x8

SET timer1 = 1000;//timer1 = 1000ms

SET digout2 = 1;//activate digital output 2
SET digout2 = 0;//deactivate digital output 2
SET pwmFreq1 = 1000; SET pwmDuty1 = 50;//1kHz, 50% duty
```

QUE Command

The format of the **QUE** command is:

```
QUE TargetOperand;
```

The meaning of command is to queue object to be transmitted or performed.

The following examples illustrate the **QUE** command usage:

```
QUE usermsg4;//queue user message 4 for transmission
QUE linFrame4;//queue LIN frame 4 for transmission
QUE frayFrame4;//queue Flexray frame 4 for transmission
QUE mfCan4;//queue CAN multi-frame 4 pre-configured resource
for transmission

QUE cl-tag2; //queue command line tagged 2 to be performed
QUE cl-num5;//queue command line number 5 to be performed
QUE cl-offset+1;//queue next command line to be performed
```

ENABLE and DISABLE Commands

The **ENABLE**/**DISABLE** commands are one operand commands used primarily to enable/disable Message Filters, Timers, logging etc.

The general format of the **ENABLE** and **DISABLE** Commands are:

ENABLE TargetOperand; and **DISABLE** TargetOperand;

The following examples illustrate of the **ENABLE**/**DISABLE** commands usage:

```
DISABLE  timer3;//disable timer 3
DISABLE filter3;//disable filter 3
ENABLE filter3;//enable filter 3
ENABLE sdLog;//enable SD card logging (open new file)
DISABLE sdLog;//disable SD card logging (close file)
ENABLE linTable1; //start LIN schedule table 1
DISABLE linTable1; //stop LIN schedule table 1
```

IF, ELSE, ELSEIF, ENDIF, ENDCL Commands

The **IF**/**ELSEIF** commands are two operand commands used to implement conditional programming statements and control program flow, whereas the **ELSE**/**ENDIF** commands are no operand commands used to implement conditional programming statements and control program flow.
ENDCL- terminates execution of current command line.
The general forms of the **IF** and **ELSEIF** Commands are:

IF/**ELSEIF** TargetOperand Operator SourceOperand;

The general forms of the **ELSE** and **ENDIF** Commands are:

ELSE; and **ENDIF;**

Conditional programming in the Netway Emulation Programming Language is very similar to conditional programming users may have encountered in other programming languages. All IF/ELSE structures must begin with an **IF** Command Statement and end with an **ENDIF** Command Statement. An **ENDIF** Command Statement, which indicates to the program the end of Command Statements dependant on the results of an IF/ELSE structure, must eventually follow every **IF** Command Statement. The simplest form of an IF/ELSE Structure in the Netway Emulation Programming Language is:
IF TargetOperand Operator SourceOperand; **CommandStatements**; **ENDIF;**
If the condition specified by an **IF** Command Statement is true, the Command Statements following that **IF** Command Statement will be executed in sequence until either an **ELSEIF**, **ELSE**, **ENDIF**, or **ENDCL** Command Statement is encountered. However, if the condition specified by an **IF** Command Statement is false, the Command Statements following that **IF** Command Statement will not be executed, and the program will skip to either the next **ELSEIF** or **ELSE** Command Statement, the Command Statement following the next **ENDIF** Command Statement, or the next Command Line in the program if **ENDCL** is encountered.

One or more **ELSEIF** Command Statements can be added following **IF** Command Statements to add more conditional operations to an IF/ELSE

structure. An IF/ELSE structure containing an **ELSEIF** Command Statement would be of the following form:

IF TargetOperand Operator SourceOperand; **CommandStatements;**
ELSEIF TargetOperand Operator SourceOperand; **CommandStatements; ENDIF;**

If the condition specified by an **IF** Command Statement is false and the condition specified by an **ELSEIF** statement is true, the Command Statements following that **ELSEIF** Command Statement will be executed in sequence until either an **ELSEIF**, **ELSE**, **ENDIF**, or **ENDCL** Command Statement is encountered. However, if the condition specified by the **ELSEIF** Command Statement is false, the Command Statements following that **ELSEF** Command Statement will not be executed, and the program will skip to either the next **ELSEIF** or **ELSE** Command Statement, the Command Statement following the next **ENDIF** Command Statement, or the next Command Line in the program if **ENDCL** is encountered.

An **ELSE** Command Statements can be added following **IF**/**ELSEIF** Command Statements to add a default conditional operation to an IF/ELSE structure. An IF/ELSE structure containing an **ELSE** Command Statement would be of the following format:

```
IF TargetOperand Operator SourceOperand;
     CommandStatements;
ELSEIF TargetOperand Operator SourceOperand;
     CommandStatements;
ELSE;
     CommandStatements;
ENDIF;
```

If an **ELSE** Command Statement is encountered (indicating all conditions specified by any previous **IF**/**ELSEIF** Command Statements are false), the Command Statements following that **ELSE** Command Statement will be executed in sequence until either an **ENDIF** or **ENDCL** Command Statement is encountered (at which point the program will skip to either the Command Statement following the **ENDIF** Command Statement if one us present, or the next Command Line).

The general form of IF/ELSE statements in the Netway Emulation Programming Language is as follows:

```
IF TargetOperand Operator SourceOperand; //IF Command Statement
```

```
     CommandStatementA1;
     CommandStatementA2;
          ...;
     CommandStatementAN;
ELSEIF TargetOperand Operator SourceOperand; //First ELSEIF Command
Statement
     CommandStatementB1;
     CommandStatementB2;
          ...;
     CommandStatementBN;
ELSEIF TargetOperand Operator SourceOperand; //2nd ELSEIF Command
Statement
     CommandStatementC1;
     CommandStatementC2;
          ...;
     CommandStatementCN;
...;
ELSEIF TargetOperand Operator SourceOperand; //Last ELSEIF Command
Statement
     CommandStatementD1;
     CommandStatementD2;
          ...;
     CommandStatementDN;
ELSE; //ELSE Command Statement
     CommandStatementE1;
     CommandStatementE2;
          ...;
     CommandStatementEN;
ENDIF; //ENDIF Command Statement
```

Note: Each Command Line can only contain up to 24 Command Statements; all **CommandStatement**s referred to above can be any valid Command Statement contained in the Netway Emulation Programming Language.

The following examples illustrate general use of the **IF**/**ELSEIF**/**ENDIF**:

```
IF x1 > 0Fh;
     SET digout4 = 0;
     SET digout5 = 0;
ENDIF;

IF x4 & x5;
     QUE usermsg4;
ELSEIF x4 ^ x5;
     QUE usermsg5;
     ENDCL;
ELSE;
     CL offset+2;
```

```
ENDIF;

IF timer2 < 4000;
     IF rxbyte2 != 50h;
          SET timer2 = 4000;
     ELSE;
          SET timer2 = 5000;
     ENDIF;
ELSEIF analog2 < 1000;
     SET analog2 = FFh;
ENDIF;
```

CL Command

The **CL** Command is a one operand command used within a Command Line to execute a different Command Line as a subroutine. When a **CL** Command is encountered during the execution of a Command Line, the Emulation Program will begin execution of the Command Statements in the Command Line specified by the operand in the **CL** Command. Once all these Command Statements have been executed, the Emulation Program will return to the original Command Line, and begin execution of the Command Statements following the **CL** Command.

The **CL** Command is of one of the following forms:

//Direct addressing where the number specified is the Index Number of a Command Line

```
CL NumericalConstant;
```

//Positive Offset addressing where `N` is the positive offset relative to the current Command Line

```
CL offset+N;
```

//Negative Offset addressing where `N` is the negative offset relative to the current Command Line

```
CL offset-N;
```

//Indirect addressing where `xN` contains the Index Number of a Command Line

```
CL xN;
```

//Indirect addressing `offset+xN -` where `xN` contains the Index Number of an offset to Command Line

```
CL offset+x;
```

//Direct addressing where `cl-TagN` contains the tag number of a Command Line (see Tag column)

```
CL cl-TagN;
```

Note: A **CL** Command must not reference the Command Line which contains it (i.e. Command Line #4 cannot contain **CL** `4;` or its' equivalent). Furthermore, the Netway6 product series allows for Emulation Programs with up to 128 Command Lines, which dictates a range of 1 to 128 for Numerical Constants and the values contained within `xN` (1024 X-Variables available), 1 to (196 – Index Number of current Command Line) (Max 195)

for `offset+N`, and 1 to (Index Number of current Command Line - 1) (Max 195) for `offset-N`.

CL Command subroutines can be nested so that either one **CL** Command subroutine calls others, or one Command Line contains multiple **CL** Command subroutines. The general form a nested **CL** Command subroutine is as follows (Note: all **CommandStatement**s referred to below can be any valid Command Statement contained in the Netway Emulation Programming Language):

Command Line #1:

```
CommandStatementA1; CommandStatementA2; ...;
CommandStatementAN;
CL 2; //First CL Command Statement
CommandStatementB1; CommandStatementB2; ...;
CommandStatementBN;
```

Command Line #2:

```
CommandStatementC1; CommandStatementC2; ...;
CommandStatementBN;
CL 3; //Second CL Command Statement
CommandStatementD1; CommandStatementD2; ...;
CommandStatementDN;
CL 4; //Third CL Command Statement
CommandStatementE1; CommandStatementE2; ...;
CommandStatementEN;
```

Command Line #3:

```
CommandStatementF1; CommandStatementF2; ...;
CommandStatementFN;
```

Command Line #4:

```
CommandStatementG1; CommandStatementG2; ...;
CommandStatementGN;
```

The program flow through the above is as follows:

1. Command Line #1 Triggered à Execute Command Statements A1-AN
2. Command Line #1: Encounter First **CL** Command à Go to Command Line #2
3. Command Line #2: Execute Command Statements C1-CN
4. Command Line #2: Encounter Second **CL** Command à Go to Command Line #3

5. Command Line #3: Execute Command Statements F1-FN à Return to Command Line #2
6. Command Line #2: Execute Command Statements D1-DN
7. Command Line #2: Encounter Third `CL` Command à Go to Command Line #4
8. Command Line #4: Execute Command Statements G1-GN à Return to Command Line #2
9. Command Line #2: Execute Command Statements E1-EN à Return to Command Line #1
10. Command Line #1: Execute Command Statements B1-BN

The following examples and explanations partially illustrate general use of the `CL` Command:

- //Execute Command Line #5; Execute Command Line #10; Execute Command Line #15

  ```
  CL 5; CL 10; CL 15;
  ```

- //Assuming the `CL` Commands are in Command Line #5: Execute Command Line #6; Execute Command Line #7; Execute Command Line #8

  ```
  CL offset+1; CL offset+2; CL offset+3;
  ```

- //Assuming the `CL` Commands are in Command Line #5: Execute Command Line #4; Execute Command Line #3; Execute Command Line #2

  ```
  CL offset-1; CL offset-2; CL offset-3;
  ```

- //Assuming `x1` = 5, `x2` = 10, and `x3` = 15: Execute Command Line #5; Execute Command Line #10; Execute Command Line #15

  ```
  CL x1; CL x2; CL x3;
  ```

- //Execute Command Line with tag 5;

  ```
  CL cl-tag5;
  ```

NETMODE Command

The *NETMODE Command* is a one operand command used to control special communication modes for several different protocols.

The general form of the *NETMODE Command* is:
NETMODE <NumericalConstant>; //Where Numerical Constant is from 1 to 54

The following examples and explanations illustrate use of the *NETMODE Command*: //Start low baud (5 bits/sec) stimulation used with the KWP2000 protocol (see nwLowBaudStimulation):
NETMODE 2; //for UART1
NETMODE 37; //for UART2
NETMODE 39; //for UART3
NETMODE 41; //for UART4
NETMODE 43; //for UART5
NETMODE 45; //for UART6
NETMODE 47; //for UART7
NETMODE 49; //for UART8
NETMODE 51; //for UART9
NETMODE 53; //for UART10
//Start Toyota KWP2000 WAKEUP pattern:
NETMODE 5; //for UART1
NETMODE 55; //for UART2
NETMODE 56; //for UART3
NETMODE 57; //for UART4
NETMODE 58; //for UART5
NETMODE 59; //for UART6
NETMODE 60; //for UART7
NETMODE 61; //for UART8
NETMODE 62; //for UART9
NETMODE 63; //for UART10
//Start fast stimulation/WAKEUP pattern used with the KWP2000 protocol (see nwFastStimulation):
NETMODE 1; //for UART1

NETMODE 36; //for UART2
NETMODE 38; //for UART3
NETMODE 40; //for UART4
NETMODE 42; //for UART5
NETMODE 44; //for UART6
NETMODE 46; //for UART7
NETMODE 48; //for UART8
NETMODE 50; //for UART9
NETMODE 52; //for UART10
//Clears network statistics (same as selecting <Monitor/Clear log and statistics> from the Menu Bar):
NETMODE 6;
//Clears log buffer :
NETMODE 54;
//Set high speed (4x-mode) for Class2 protocol (see nwSetClass2SpeedMode):
NETMODE 7;
//Reset Class2 protocol to normal speed (1x-mode) (see nwSetClass2SpeedMode):
NETMODE 8;
//Set high speed (83.3 kbit) for SCP monitoring chip:
NETMODE 64;
//Set normal speed (41.6 kbit) for SCP monitoring chip:
NETMODE 65;
//Set two least significant digits of the Netway device’s serial number as a node address:
NETMODE 9;
//Start CAN histogram statistics:
NETMODE 10;
//Stop CAN histogram statistics:
NETMODE 11;
//Start logging stuffing, format, crc, and bit CAN errors as special messages:
NETMODE 12;
//Stop logging stuffing, format, crc, and bit CAN errors as special messages:
NETMODE 13;
//Set LIN slave wake-up signal:

NETMODE 14;
//Set wake-up signal level (~11V) for Single Wire CAN hardware channel #2 bus (see nwSetGMLanWakeUpLevel):
NETMODE 3; //for CAN1
NETMODE 15; //for CAN2
NETMODE 17; //for CAN3
NETMODE 19; //for CAN4
NETMODE 21; //for CAN5
NETMODE 23; //for CAN6
NETMODE 25; //for CAN7
NETMODE 27; //for CAN8
NETMODE 29; //for CAN9
NETMODE 31; //for CAN10
//Reset CAN bus to normal mode signal level (see also nwSetGMLanNormalLevel):
NETMODE 4 ;//for CAN1
NETMODE 16; //for CAN2
NETMODE 18; //for CAN3
NETMODE 20; //for CAN4
NETMODE 22; //for CAN5
NETMODE 24; //for CAN6
NETMODE 26; //for CAN7
NETMODE 28; //for CAN8
NETMODE 30; //for CAN9
NETMODE 32; //for CAN10
//Transmit a Break Signal over Class2 hardware channel #1 (see nwSendClass2BreakSignal):
NETMODE 33;
//Transmit a Break Signal over Class2 hardware channel #2:
NETMODE 34;
//Transmit a Break Signal over Class2 hardware channel #3:
NETMODE 35;

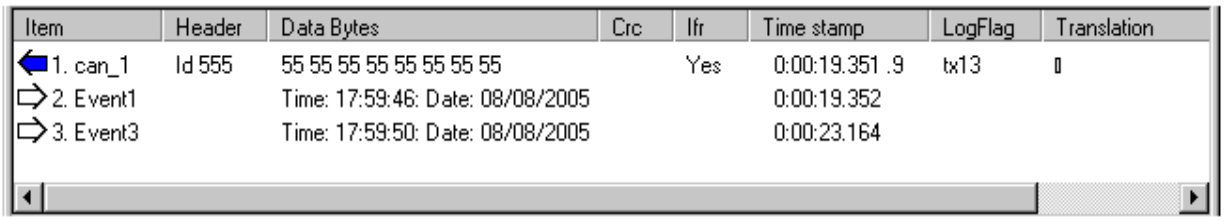
LOGEVNT Command

The **LOGEVNT** Command is a no operand command used to create a time-stamped record of the current event, and save that record in a Netway Device's internal log.
The general form of the **LOGEVNT** Command is:

```
LOGEVNT;
```

The messages pictured below can be observed in the Dynamic Monitor Area of the *Netway Application Program* after the following Command Statements are executed (assume User Message #3 is defined for CAN hardware channel #1 as 555 55 55 55 55 55 55 55):

```
QUE usermsg3; (Log Item #1) LOGEVNT; (Log Item #2)
SET x1 = 99h; LOGEVNT; (Log Item #3)
```



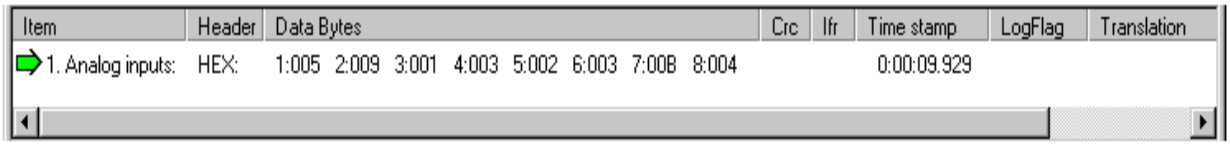
Item	Header	Data Bytes	Crc	Ifr	Time stamp	LogFlag	Translation
1. can_1	Id 555	55 55 55 55 55 55 55 55		Yes	0:00:19.351 .9	tx13	
2. Event1		Time: 17:59:46: Date: 08/08/2005			0:00:19.352		
3. Event3		Time: 17:59:50: Date: 08/08/2005			0:00:23.164		

LOGANLG Command

The **LOGANLG** Command is a no operand command used to create a time-stamped record of the current analog values of all the Digital/Analog Inputs, and send that record in a Netway Device's internal log.
The general form of the **LOGANLG** Command is:

```
LOGANLG;
```

The message pictured below can be observed in the Dynamic Monitor Area of the *Netway Application Program* following execution of the above Command Statement:



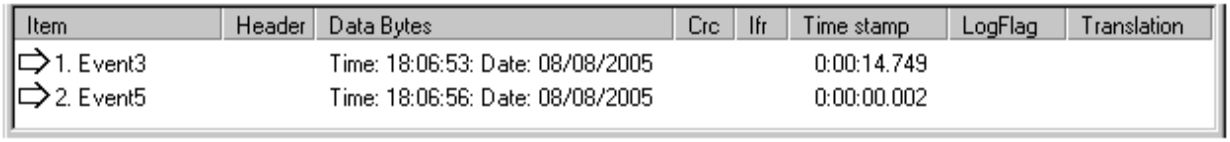
RESETTMR Command

The **RESETTMR** Command is a no operand command used to reset the Log Timer to zero. Upon power-up of a Netway device the Log Timer is reset to zero, and then incremented every millisecond. Each item logged by a Netway device is entered in the log buffer with a time stamp derived from the Log Timer. When an Emulation Program encounters the **RESETTMR** Command Statement, the Log Timer is reset to zero.
The general form of the **RESETTMR** Command is:

```
RESETTMR;
```

The effect of the **RESETTMR** Command (pictured below) can be verified by observing the Time Stamp Column of the Dynamic Monitor Area and executing the following Command Statements:

LOGEVNT; (Log Item #2) **RESETTMR; LOGEVNT;** (Log Item #3)



Item	Header	Data Bytes	Crc	Ifr	Time stamp	LogFlag	Translation
1. Event3		Time: 18:06:53: Date: 08/08/2005			0:00:14.749		
2. Event5		Time: 18:06:56: Date: 08/08/2005			0:00:00.002		

LOOP command

The **LOOP** Command is a one operand command perform variety of loop operations.
The general form of the **SET** Command is:

```
LOOP SourceOperand;
```

The following examples and explanations partially illustrate general usage of the **LOOP** Command:

- //In this example fourteen variables y20 - y33 are filled with values 21 - 34

SET y1 = 20;
LOOP Eh; //loop 14 times
SET y1 + 1; //increment y1; **SET** indY1 = y1; //assign value to selected Y-Variable
LOOPEND ;

- //In this example we want to find Y-variable that matches a specific value 1234 hex

SET y1 = 20;
SET x1 = 0;
SET x2 = 64;
LOOP x2; //loop through y20-y83
// CL cl-tag1; //optionally we can call any command line within the loop
SET y1 + 1;
IF indY1 == 1234h;
SET x1 = 1; //found!
LOOPBRK ;
ENDIF ;
LOOPEND ;

Using Operands

A Command Statement is expressed in one of the following forms:

- No operand: **`Command`**
- One Operand: **`Command TargetOperand`**
- Two Operands: **`Command TargetOperand`** Operator `SourceOperand`

The following chart contains all of the operands available in the Emulation Programming Language (rows), and indicates which Commands (columns) they can be used as the target operand for:

	SET	QUE	IF/ELSEIF	ENABLE	DISABLE	CL	NETMODE
xN	X		X			X	
yN	X		X				
indXn	X						
indYn	X						
zN	X		X				
xbit, ybit, rxbit,umsgbit	X		X				
timerN	X		X		X		
netmsgN	X	X					
usermsgN	X	X		X	X		
umsgXN	X	X					
ncfXN	X	X					
canAobjN	X	X		X	X		
canBobjN	X	X		X	X		
blockN	X	X					
digoutN	X		X				
pwmFreqN	X						
pwmDutyN	X						
ifrdataN	X						
txgrpN	X	X		X	X		
rxbyteN			X				
analogN	X		X				
pwmFreqN			X				
pwmDutyN			X				
diginpN			X				
delayN	X	X					
filterN				X	X		
log				X	X		
replayN				X	X		
emulation					X		
debounce				X	X		
offset+N						X	
offset-N						X	
linTableN				X	X		
fractionN							
uartType	X						
uartTxgap	X						
uartBitrate	X						
canBitrate	X						
kwParm	X						

rtcN	X						
Numerical Constant						X	X
	SET	QUE	IF/ELSEIF	ENABLE	DISABLE	CL	NETMODE

Note: A detailed description of every operand available in the Emulation Programming Language can be found in the by following the desired link above.

The following chart contains all of the target operands available in the Emulation Programming Language (rows), and indicates which source operands (columns) they can be used with:

	xN	yN	indXn	indYn	zN	timerN	usermsgN	blockN	digoutN	rxbyteN	analogN	pwmF
xN	X	X			X			X	X	X	X	X
yN	X	X		X	X	X		X		X	X	X
indXn	X		X									
indYn		X										
zN	X	X			X	X		X		X	X	X
timerN	X	X			X					X		
netmsgN	X	X			X			X		X	X	
usermsgN	X	X			X			X		X	X	
umsgXN	X	X			X			X		X	X	
ncfXN	X	X			X			X		X	X	
canAobjN	X	X			X			X		X	X	
canBobjN	X	X			X			X		X	X	
blockN	X	X			X			X*		X		
digoutN	X	X			X					X		
freq(Hz)N		X			X							
duty(%)N	X									X	X	
ifrdataN	X									X	X	
txgrpN	X	X					X					
rxbyteN	X											
analogN	X											
pwmFreqN	X	X			X			X	X	X	X	X
pwmDutyN	X	X			X			X	X	X	X	X
diginpN	X											

* Only commands that use block1 or block7 as the target operand can use blockN as a source operand.

Note: delayN, filterN, log, replayN, emulation, debounce, linTableN, offset+N, and offset-N are only associated with one operand commands, and are therefore not included in the above chart. Numerical Constants are not included in the target operand list because they can only be a target operand in conjunction with the **NETMODE** command.

Operators

The following contains an explanation of all the operators available for use with the **SET** command: (assume x1 = 5)

Symbol	Name	Functionality	Example Command	Result
=	Equals Operator	Set the target operand equal to the source operand	**SET** x1 = 45; //x1 ß 45	x1 = 45
+	Plus Operator	Set the target operand equal to the target operand plus the source operand	**SET** x1 + 45; //x1 ß x1 + 45	x1 = 50
-	Minus Operator	Set the target operand equal to the target operand minus the source operand	**SET** x1 - 45; //x1 ß x1 45	x1 = 40
*	Times Operator	Set the target operand equal to the target operand times the source operand	**SET** x1 * 45; //x1 ß x1 * 45	x1 = 225
/	Divided-By Operator	Set the target operand equal to the target operand divided by the source operand	**SET** x1 / 5; //x1 ß x1 / 5	x1 = 1
|	OR Operator	Set the target operand equal to the target operand ORed with the source operand	**SET** x1 | 45h; //x1 ß x1 OR $45	x1 = $45
&	AND Operator	Set the target operand equal to the target operand ANDed with the source operand	**SET** x1 & 45h; //x1 ß x1 AND $45	x1 = $5
^	XOR Operator	Set the target operand equal to the target operand XORed with the source operand	**SET** x1 ^ 45h; //x1 ß x1 XOR $45	x1 = $40
~	Invert (NOT) Operator	Set the target operand equal to NOT the source operand	**SET** x1 ~ 45h; //x1 ß $45	x1 = $BA
>>	Right-Shift Operator	Shift the bits in the target operand to the right source operand places	**SET** x1 >> 2; //x1 ß x1 right-shift 2-bits	x1 = $01
<<	Left-Shift Operator	Shift the bits in the target operand to the left source operand places	**SET** x1 << 4; //x1 ß x1 left-shift 4-bits	x1 = $50

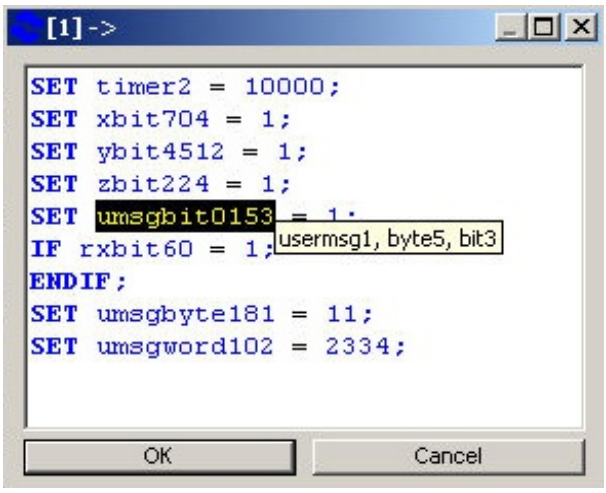
The following contains an explanation of all the operators available for use with the **IF**/**ELSEIF** commands:

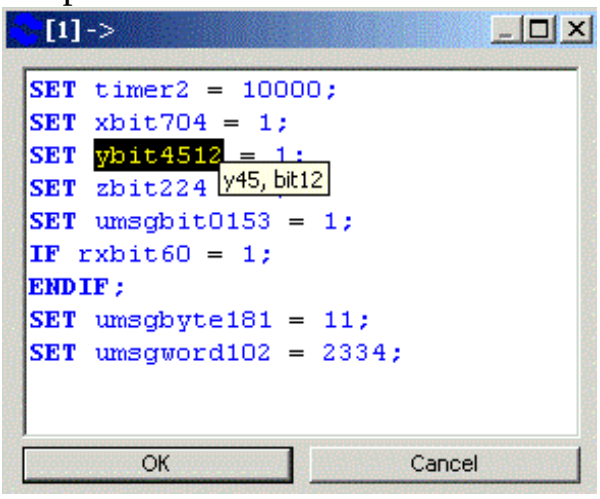
Symbol	Name	Function	Example	Result	
				x1 = $05	x1 = $48
==	Equivalence operator	If the target operand is equal to the source	**IF** x1 == 48h; //x1 is 48??	FALSE	TRUE

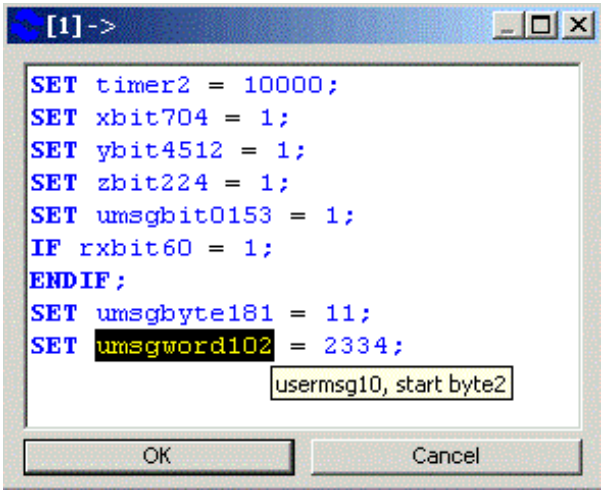
		operand then the statement is true			
!=	Nonequivalence operator	If the target operand is not equal to the source operand then the statement is true	`IF x1 != 48h;` //`x1` is anything but 48??	TRUE	FALSE
<	Less-Than Operator	If the target operand is less than the source operand then the statement is true	`IF x1 < 47h;` //`x1` is less than 47??	TRUE	FALSE
>	Greater-Than operator	If the target operand is greater than the source operand then the statement is true	`IF x1 > 47h;` //`x1` is greater than 47??	FALSE	TRUE
&	AND Operator	If the result of a logical AND between target operand and source operand is not equal to zero then the result is true	`IF x1 & 48h;` //`x1` AND 48 is not zero??	FALSE	TRUE
^	XOR Operator	If the result of a logical XOR between target operand and source operand is not equal to zero then the result is true	`IF x1 ^ 48h;` //`x1` XOR 48 is not zero??	TRUE	FALSE

Manipulating with bits, bytes, words...

Due to the way some operands are encoded, the Command Line Editor (see Editing Command Lines) displays some hints to ease the programming. When dealing with bit-wise operands user can select (highlight) them and the hint will be displayed as shown in the pictures below:







Netway language offers number of operands for convenient data manipulation of different sizes: bits, bytes, word, etc. The following are a list of operands described in details in this chapter:
Bit-wise operands: xbit, ybit, zbit, rxbit, umsgbit **(SET, IF, ELSEIF)**
Byte-wise operands: rxbyte, rxdata, umsgbyte **(SET, IF, ELSEIF)**
Word-wise operands: umsgword **(SET, IF, ELSEIF)**
Special operands: bitsize, bytesize **(ENABLE)**

Format of xbit,ybit,zbit operands: xbitNNNNN, ybitNNNN, zbitNNNN:
NNNNN/100 (hundreds) -represent variable number
Two least significant decimal digits represent start bit index (0-7 for X, 0-15 for Y and 0-31 for Z).
Examples:
xbit704 - X-variable 7, start bit position 4
ybit4512 - Y-variable 45 , start bit position 12
zbit224 - Z-variable 2, start bit position 24

Format of umsgbit operand: umsgbitNNN:
number NNN/100 (hundreds) -represent user message number second from the left decimal digit (tens) - represents byte index within the message, least significant digit represents bit index within the byte
ENABLE bitsize2;
For example SET umsgbit453 = ybit1003; // sets byte 5 of user message 4, start bit position 3, with two bits Suppose user message 1 is defined as: 301 00 00 00 00 00 00 00 00
The following command line:
SET umsgbit0153 = 1;QUE usermsg1; //user message 1, byte pos. 5, bit 3 will result in transmitting message:
301 00 00 00 00 00 08 00 00

Format of rxbit operand: rxbitNN:
number NN/10 (tens) -represent message byte index (0-7) Least significant decimal digit represent bit index within the byte (0-7)
rxbit60 means byte 6 of received message, start bit 0;

Example 1:

ENABLE bitsize4;
SET xbit104 = rxbit60; // sets x1, start bit 4, with 4 bits from rxbyte 6, start bit 0

Example 2:

Suppose that user message 4 is defined as:
304 00 00 00 00 00 00 00 00

then can1flt3 is set to can ID $123 and command line 4 is triggered by can1flt3:
ENABLE bitsize4;
SET umsgbit403 = rxbit0073;
QUE usermsg4;
If incoming message 123 FF FF FF FF FF FF FF **5A** is detected, the result will be:
304 **0A** 00 00 00 00 00 00 00

Example 3:

The same as example 2, but command line 4 is defined as: ENABLE bitsize4;
IF rxbit73 & 11h;
    ENABLE bitsize4;
    SET umsgbit403 = 0Fh;
ELSE;
    ENABLE bitsize4;
    SET umsgbit403 = 0Ah;
ENDIF;
QUE usermsg4;
If incoming message: 123 00 00 00 00 00 00 00 66 , the resulting message will be:
304 0A 00 00 00 00 00 00 00
If incoming message: 123 00 00 00 00 00 00 00 77 , the resulting message will be:
304 0F 00 00 00 00 00 00 00

Format of bitsize operand: bitsizeN:
This operand is used with ENABLE command, N - represents number of bits affected by the next bitwise command.
For example ENABLE bitsize4;
NOTE: This command enables number of affected bits for the next command only, if not present only one bit will be affected!

Format of bytesize operand: bytesizeN:

This operand is used with ENABLE command, N - represents number of bytes affected by the next command.
For example:
ENABLE bytesize2; SET umsgbyte181 = z2; NOTE: This ENABLE bytesizeN; command enables number of affected bytes for the next command only, if not present only one byte will be affected!

Format of umsgbyte operand: umsgbyteNNN:
NNN/10 (tens) - represent user message number
Least significant decimal digit represents byte index within the message.
For example: ENABLE bytesize2; SET umsgbyte181 = z2; QUE usermsg18;
If original user message 18 is defined as can1: 00000101 01 02 03 04 05 06 07 08, and z2 = 0x11223344, resulting message will be: 00000101 01 ***33 44*** *04 05 06 07 08*
bytes at position 1 and 2 affected by the command line.

Format of umsgword operand: umsgwordNNN:
NNN/10 (tens) - represent user message number
Least significant decimal digit represents word index within the message.
For example: user message 10 is defined as: 311 10 20 30 40 50 60 70 80
SET umsgword100 = 1122h;
SET y9 = 3344h;
SET umsgword102 = y9;
SET y9 = 5566h;
SET umsgword104 = y9;
SET y9 = 7788h;
SET umsgword106 = y9;
QUE usermsg10;
Resulting message will be: 311 **11 22 33 44 55 66 77 88**

xN, yN, zN, and blockN

xN, where N is from 0 to 256: Used to reference the 256 1-byte data registers known as the X-Variables. xN operands can be used as either source or target operands, however if the variable is not initialized before use as a source operand then the result will be unpredictable. The 8-bit X-Variable size dictates a numerical range of 0-FF hexadecimal or 0-255 decimal. A special reference to the X-Variable, x0, is included in the Emulation Programming Language for the purpose of setting all 256 X-Variables to one value with a minimum amount of Command Statements. For example: the Command Statement **SET** x0 = 00h; will set all 256 X-Variables to zero; this type of Command is usually found as an initialization at the beginning of Emulation Programs. xN operands can also be employed in conditional programming commands (**IF**/**ELSEIF**).
Examples:

```
SET x0 = 00h;
        //Set the value contained in x1-x256 equal to zero
SET x50 + x200;
        //Set the value contained in x50 equal to the value contained in x200
SET x50 + 5;
        //Set the value contained in x50 equal to the value contained in x50 plus 5
SET x10 – y1;
        //Set x10 = x10 – least significant byte of y1
CL x74;
        //Execute the command with index number equal to the value contained in x74
```

yN, where N is from 0 to 64: Used to reference the 64 2-byte data registers known as the Y-Variables. yN operands can be used as either source or target operands, however if the variable is not initialized before use as a source operand then the result will be unpredictable. The 16-bit Y-Variable size dictates a numerical range of 0-FFFF hexadecimal or 0-65535 decimal. A special reference to the Y-Variable, y0, is included in the Emulation Programming Language for the purpose of setting all 64 Y-Variables to one value with a minimum amount of Command Statements. For example: the

Command Statement `SET y0 = 0000h;` will set all 64 Y-Variables to zero; this type of Command is usually found as an initialization at the beginning of Emulation Programs. `yN` operands can also be employed in conditional programming commands (**`IF`**/**`ELSEIF`**).
Examples:

```
SET y0 = 9999;
        //Set the value contained in y1-y64 equal to 9999 (decimal)
SET y9 = y19;
        //Set the value contained in y9 equal to the value contained in
y19
SET y50 / y5;
        //Set the value contained in y50 equal to the value contained
in y50 divided by y5
```

`zN`, where `N` is from 0 to 16: Used to reference the 8 4-byte data registers known as the Z-Variables. `zN` operands can be used as either source or target operands, however if the variable is not initialized before use as a source operand then the result will be unpredictable. The 32-bit Z-Variable size dictates a numerical range of 0-FFFFFFFF hexadecimal or 0-4294967295 decimal, however data must be entered into the Z-Variables 4 bytes at a time. `zN` operands can also be employed in conditional programming commands (**`IF`**/**`ELSEIF`**).
Examples:

```
SET z6 = 1234h; SET z6 << 16; SET z6 | 5678h;
        //Set the value contained in z6 equal to $00001234; Set the
value contained in z6 equal to $12340000; Set the value contained in z6
equal to $12345678
SET z1 = z11;
        //Set the value contained in z1 equal to the value contained in
z11
SET z5 * x2;
        //Set the value contained in z5 equal to the value contained in
z5 times the value contained in x2
```

Note: The Z-Variables are primarily intended for internal system calculations, and consequently users should avoid using the Z-Variables in Emulation Programs when possible.

`blockN`, where `N` is from 1 to 12: Used to define and control segments of the Block Memory. `blockN` operands can be used as either target or source operands, however if the memory is not initialized before use as a source operand then the result will be unpredictable. Refer to the Block Memory section for a detailed description of all the `blockN` operands.

timerN, delayN, linTableN, and ifrdataN

timerN, where N is from 0 to 32: Used to reference the 32 2-byte timers. timerN operands can be used as either source or target operands, however if the timer is not initialized before use as a source operand then the result will be unpredictable. The value contained in all the timerN operands is decremented by one every 1 millisecond until that value reaches zero, and the transition of a timerN operand's value from 1 to 0 is an event that can be used to Trigger the execution of a Command Line. When a timerN operand contains a nonzero value, that timer is said to be enabled or active. A timer is said to be passive (i.e. not modified by any internal process) when its' value reaches zero or if it is disabled. On power-up, all the timerN operands are initialized to zero (i.e. all timers are passive). The 16-bit timer size dictates a numerical range of 0-FFFF hexadecimal or 0-65535 decimal, and this governs a periodic range of up to 65535ms or 65.535 seconds. A special reference to the timerN operands, timer0, is included in the Emulation Programming Language for the purpose of setting all 32 timers to one value with a minimum amount of Command Statements. For example: the Command Statement `SET timer0 = 1000;` will set all 32 timers to one second; this type of Command is usually found as an initialization at the beginning of Emulation Programs. timerN operands can also be employed in conditional programming commands (**IF**/**ELSEIF**). Examples:

- `SET timer4 = 65535;`
  //Set the value contained in timer4 equal to 65535ms
- `SET y1 = timer31;`
  //Set the value contained in y1 equal to the current value contained in timer31
- `SET timer15 + 11;`
  //Set the value contained in timer15 equal to the value contained in timer15 + 11ms
- `DISABLE timer3;`
  //Set the value contained in timer3 equal to 0 (make timer passive)

delayN, where N is from 1 to 65535 (**QUE**)/1 to 32 (**SET**): Used with the **QUE** Command, the delayN operand is used to put user defined time gaps

between the completion of the transmission of any message previously in the transmission queue and the transmission of the next message that will be put into the transmission queue. It is equivalent to sending timed delays to a Netway device's transmit queue. The length of the delay is expressed in milliseconds and determined by the value of `N` (maximum: 65535 milliseconds). Used with the **`SET`** Command, the `delayN` operand is used to put user defined time gaps into the program flow of Command Statement execution, and can be useful for creating periodic events. It is equivalent to making the program 'hang' for a specified interval. The **`SET`** `delayN` command uses one of the 32 2-byte timers as a resource for the time gap, where `N` specifies the specific timer to use. The length of the delay is expressed in milliseconds and determined by the value of the associated source operand (maximum: 65535 milliseconds).

Note: Using the **`SET`** `delayN = Number;` command causes the equivalent to executing **`SET`** `timerN = Number;` to occur, and will therefore reset `timerN`. As a result users must avoid using a **`SET`** `delayN` command statement if `timerN` is already used for Emulation Program control. Examples:

- **`QUE`** `usermsg1;` **`QUE`** `delay5000;` **`QUE`** `netmsg2;`
  //Transmit User Message #1, wait 5 seconds, transmit NCF message #2
- **`QUE`** `delay1000;` **`QUE`** `canAobj7;`
  //Wait 1 second, transmit CAN message #7
- **`CommandStatementA;`** **`SET`** `delay4 = 1000;` **`CommandStatementB;`**
  //Execute CommandStatementA; wait 1 second; execute CommandStatementB

Note: **`QUE`** `delayN` is not compatible with CAN hardware channels.

`linTableN`, where `N` is from 0 to 8: Special operand valid only with the LIN Protocol, and used with the **`ENABLE`**/**`DISABLE`** commands to start/stop up to 8 individual schedule tables. A special `linTableN` operand `linTable0` is included to disable the active schedule table. Starting a schedule table while another one is active will automatically stop the active table before starting the new one. Examples:

- **`ENABLE`** `linTable1;` **`DISABLE`** `linTable1;`
  //Start schedule table #1; Stop schedule table #1
- **`ENABLE`** `linTable6;` **`ENABLE`** `linTable4;` **`DISABLE`** `linTable0;`

//Start schedule table #6; Stop schedule table #6 and start schedule table #4; Stop active schedule table (#4)

`ifrdataN`, where `N` is from 1 to 8: Used only to set the in-frame data for function-read messages in the SCP protocol. See the Node Configuration Button section for more information. `ifrdataN` operands can be assigned 8-bit numerical values, which dictates a numerical range from $0-$FF (0-255 decimal), and therefore can store up to 8 message bytes for the SCP protocol. `ifrdataN` operands can only be used as target operands, and data from X-Variables, analog inputs, and received messages can be sent to an `ifrdataN` operand with the **`SET`** command. Examples: (assume `x100` = $FA, `analog4` = 53, and the message 11 22 33 44 was just received)

- **`SET`** `ifrdata1 = x100;` **`SET`** `ifrdata7 = analog4;`
  //Set in frame data byte #1 equal to $FA; set in frame data byte #7 equal to $53
- **`SET`** `ifrdata4 = rxbyte2;` **`SET`** `ifrdata6 = BBh;`
  //Set in frame data byte #4 equal to $22 Set in frame data byte #6 equal to $BB

usermsgN, canAobjN, and canBobjN

usermsgN, where N is from 1 to 255:
Used to reference the up to 255 messages defined in the User Message File associated with a given Emulation Program. See the User Messages Tab-Window section for more information.
usermsgN operands can be used as either target or source operands, however if the message is not defined before use as an operand then the result will be unpredictable. Data from memory (Block Memory, X-, Y-, and Z-Variables), received messages, and analog inputs can be put into the undefined bytes (represented by XX) of a usermsgN operand with the SET command. Data from the source operand specified will be placed into the first undefined message byte, and then data from subsequent sequential source locations will be filled in the remaining undefined message bytes until all the undefined bytes have been defined or no more source locations are available. This operation can be preformed multiple times in an *Emulation Program* to allow users to redefine previously defined undefined bytes, and the data placed into an undefined message byte will remain there until the *Netway device* is powered off and the undefined message bytes return to being undefined. usermsgN operands can also be used with the QUE command to place copies of messages (as modified by any previous commands) into the Netway device's transmit queue. This action will cause the message referenced by the usermsgN operand to be transmitted by the Netway device on the protocol specified in the Type Column of the User Messages Tab-Window.
Examples: (assume *User Message* #4 was defined as 57 18 65 XX XX XX, yN = NNNN, and the message 11 22 33 44 was just received (not on CAN1)) SET usermsg4 = rxbyte1; DISABLE usermsg4; QUE usermsg4; // Set the undefined bytes of User Message #4 equal to 22 33 44 and DO NOT transmit 57 18 65 22 33 44

> SET usermsg4 = y4; ENABLE usermsg4; QUE usermsg4; // Set the undefined bytes of User Message #4 equal to 44 44 55 and transmit 57 18 65 44 44 55

canAobjN, where N is from 1 to the index number of the last message defined in the CAN Configuration Tab-Window for hardware channel 1:

Used to reference one of the messages defined for hardware channel #1 in the CAN Message File associated with a given *Emulation Program*. See the CAN Configuration Tab-Window section for more information. canAobjN operands can only reference messages defined on CAN hardware channel #1; in order to reference CAN messages defined for hardware channels higher than 2, users can additionally define the message in the *User Messages Tab-Window* and employ usermsgN operands. canAobjN operands can only be used as target operands, however if the message is not defined before use as a target operand then the result will be unpredictable. Data from memory (Block Memory, X-, Y-, and Z-Variables), received messages, and analog inputs can be put into the undefined bytes (represented by XX) of a canAobjN operand with the SET command. Data from the source operand specified will be placed into the first undefined message byte, and then data from subsequent sequential source locations will be filled in the remaining undefined message bytes until all the undefined bytes have been defined or no more source locations are available. This operation can be preformed multiple times in an *Emulation Program* to allow users to redefine previously defined undefined bytes, and the data placed into an undefined message byte will remain there until the Netway device is powered off and the undefined message bytes return to being undefined. canAobjN operands can also be used with the QUE command to place copies of messages (as modified by any previous commands) into the Netway device's transmit queue. The ENABLE/DISABLE commands can also be used with canAobjN operands to allow/disallow CAN Messages from being transmitted; a Netway device will not transmit a CAN Message that has been disabled.
Examples: (assume CAN1 Message #4 was defined as 057 18 XX XX, xN = NN, and the message 11 22 33 44 was just received (not on CAN1)) SET canAobj4 = rxbyte1; DISABLE canAobj4; QUE canAobj4; // Set the undefined bytes of CAN1 Message #4 equal to 22 33 and DO NOT transmit 057 18 22 33

SET canAobj4 = x6; ENABLE canAobj4; QUE canAobj4; // Set the undefined bytes of CAN1 Message #4 equal to 66 77 and transmit 057 18 66 77

canBobjN, where N is from 1 to the index number of the last message defined in the CAN Configuration Tab-Window for hardware channel 2:

Used to reference one of the messages defined for hardware channel #2 in the *CAN Message File* associated with a given *Emulation Program*. See the CAN Configuration Tab-Window section for more information. canBobjN operands can only reference messages defined on CAN hardware channel #2; in order to reference CAN messages defined for hardware channels higher than 2, users can additionally define the message in the *User Messages Tab-Window* and usermsgN operands. canBobjN operands can only be used as target operands, however if the message is not defined before use as a target operand then the result will be unpredictable. Data from memory (Block Memory, X-, Y-, and Z-Variables), received messages, and analog inputs can be put into the undefined bytes (represented by XX) of a canBobjN operand with the SET command. Data from the source operand specified will be placed into the first undefined message byte, and then data from subsequent sequential source locations will be filled in the remaining undefined message bytes until all the undefined bytes have been defined or no more source locations are available. This operation can be preformed multiple times in an Emulation Program to allow users to redefine previously defined undefined bytes, and the data placed into an undefined message byte will remain there until the Netway device is powered off and the undefined message bytes return to being undefined. canBobjN operands can also be used with the QUE command to place copies of messages (as modified by any previous commands) into the Netway device's transmit queue. The ENABLE/DISABLE commands can also be used with canBobjN operands to allow/disallow CAN Messages from being transmitted; a Netway device will not transmit a CAN Message that has been disabled.

Examples: (assume CAN2 Message #4 was defined as 057 18 XX XX, xN = NN, and the message 11 22 33 44 was just received (not on CAN2)) SET canBobj4 = rxbyte1; DISABLE canBobj4; QUE canBobj4; // Set the undefined bytes of CAN2 Message #4 equal to 22 33 and DO NOT transmit 057 18 22 33

SET canBobj4 = x6; ENABLE canBobj4; QUE canBobj4; // Set the undefined bytes of CAN2 Message #4 equal to 66 77 and transmit 057 18 66 77

umsgXN, ncfXN, and txgrpN

umsgXN, where N is from 1 to 256: Similar to usermsgN operands, umsgXN operands are used to reference the up to 255 messages defined in the User Message File associated with a given Emulation Program with an offset value from any of the X-Variables. See the User Messages Tab-Window and X-Variables sections for more information. The XN part of the operand refers to an X-Variable location, and the value contained in that variable becomes an offset (valid offset values 0-254) for a User Message index number. Therefore using umsgXN references the User Message defined in the User Messages Tab-Window at index number [XN], and hence umsgXN references User Message #1 if [XN] = 0, umsgXN references User Message #2 if [XN] = 1, … , and umsgXN references User Message #255 if [XN] = 254. umsgXN operands can only be used as target operands, however if the message is not defined before use as a target operand then the result will be unpredictable. Data from memory (Block Memory, X-, Y-, and Z-Variables), received messages, and analog inputs can be put into the undefined bytes (represented by XX) of a umsgXN operand with the **SET** command. Data from the source operand specified will be placed into the first undefined message byte, and then data from subsequent sequential source locations will be filled in the remaining undefined message bytes until all the undefined bytes have been defined or no more source locations are available. This operation can be preformed multiple times in an Emulation Program to allow users to redefine previously defined undefined bytes, and the data placed into an undefined message byte will remain there until the Netway device is powered off and the undefined message bytes return to being undefined. umsgXN operands can also be used with the **QUE** command to place copies of messages (as modified by any previous commands) into the Netway device's transmit queue. This action will cause the message referenced by the umsgXN operand to be transmitted by the Netway device on the protocol specified in the Type Column of the User Messages Tab-Window.

Examples: (assume User Message #N was defined as NN XX XX XX, xN = 0N, z3 = $12345678, analogN = NN, and the message 11 22 33 44 55 was just received (not on CAN1))

- **SET** umsgX7 = rxbyte2; **QUE** umsgX7;

//Set the undefined bytes of User Message #8 equal to 33 44 55 and transmit 88 33 44 55

- `SET umsgX3 = z3; QUE umsgX3;`
  //Set the undefined bytes of User Message #4 equal to 12 23 56 and transmit 44 12 34 56
- `SET umsgX1 = analog4; QUE umsgX1;`
  //Set the undefined bytes of User Message #2 equal to 44 55 66 and transmit 22 44 55 66

`ncfXN`, where `N` is from 1 to 256: Similar to `netmsgN` operands, `ncfXN` operands are used to reference the up to 256 messages defined in the Network Configuration File (NCF) associated with a given Emulation Program with an offset value from an X-Variable. See the NCF Messages Tab-Window and X-Variables sections for more information. The `XN` part of the operand refers to an X-Variable location, and the value contained in that variable becomes an offset (valid values 0-255) for a NCF Message index number. Therefore using `ncfXN` references the NCF Message defined in the NCF Messages Tab-Window at index number [`XN`], and hence `ncfXN` references NCF Message #1 if [`XN`] = 0, `ncfXN` references NCF Message #2 if [`XN`] = 1, … , and `ncfXN` references NCF Message #256 if [`XN`] = 255. `ncfXN` operands can only be used as target operands, however if the message is not defined before use as a target operand then the result will be unpredictable. Data from memory (Block Memory, X-, Y-, and Z-Variables), received messages, and analog inputs can be put into the undefined bytes (represented by XX) of a `ncfXN` operand with the **`SET`** command. Data from the source operand specified will be placed into the first undefined message byte, and then data from subsequent sequential source locations will be filled in the remaining undefined message bytes until all the undefined bytes have been defined or no more source locations are available. This operation can be preformed multiple times in an Emulation Program to allow users to redefine previously defined undefined bytes, and the data placed into an undefined message byte will remain there until the Netway device is powered off and the undefined message bytes return to being undefined. `ncfXN` operands can also be used with the **`QUE`** command to place copies of messages (as modified by any previous commands) into the Netway device’s transmit queue. This action will cause the message referenced by the `ncfXN` operand to be transmitted by the

Netway device on the protocol specified in the Type Column of the NCF Messages Tab-Window.
Examples: (assume NCF Message #N was defined as NN XX XX XX, `xN` = 0N, `y23` = $1234, `y24` = $5678, `analogN` = NN, and the message 11 22 33 44 55 was just received (not on CAN1))

- `SET ncfX4 = rxbyte2; QUE ncfX4;`
  //Set the undefined bytes of NCF Message #5 equal to 11 22 33 and transmit 55 33 44 55
- `SET ncfX2 = y23; QUE ncfX2;`
  //Set the undefined bytes of NCF Message #3 equal to 12 34 56 and transmit 33 12 34 56
- `SET ncfX8 = analog2; QUE ncfX8;`
  //Set the undefined bytes of NCF Message #9 equal to 22 33 44 and transmit 99 22 33 44

`txgrpN`, where `N` is from 1 to 8: Used to reference the an Emulation Program's transmit groups. An Emulation Program can have up to 8 transmit groups defined, and each transmit group consists of up to 8 User Messages with **Delays** (**Delay Column** in User Message Tab-Window determines gap before message transmission) and a periodic rate. One can place User Messages with **Delays** into transmit group #N by using the **`SET`** command with `txgrpN` as the target operand and `usermsgN` as the source operand. One can set the 16-bit periodic rate of the transmit group (numerical range of 0-FFFF hexadecimal or 0-65535 decimal dictates periodic range of up to 65535ms or 65.535 seconds) by using the **`SET`** command with `txgrpN` as the target operand and either `xN`, `yN`, or a numerical constant as the source operand. Once a transmit group is activated (**`QUE`** or **`ENABLE`** commands), a Netway device will transmit the up to 8 User Messages contained in that group separated by the defined **Delays**, wait the amount of time dictated by the periodic rate, and then transmit the up to 8 User Messages contained in that group separated by the defined **Delays** again. This cycle will continue until the transmit group is deactivated (**`DISABLE`** command).
Examples: (assume `y30` = 1000 and `usermsgN` = NN NN NN with **Delay** = N0)

- `SET txgrp3 = y30; SET txgrp3 = usermsg4; SET txgrp3 = usermsg5;`

//Set the periodic rate of transmit group #3 equal to 1000ms and put User Message #4 and User Message #5 into transmit group #3

- `SET txgrp6 = 500; SET txgrp6 = usermsg6; SET txgrp6 = usermsg2;`
  ////Set the periodic rate of transmit group #6 equal to 500ms and put User Message #6 and User Message #2 into transmit group #6
- `QUE txgrp3; ENABLE txgrp6;`
  //Transmit 44 44 44, wait 50ms, transmit 55 55 55, wait 1s, transmit 44 44 44, wait 50ms, transmit 55 55 55…; Transmit 66 66 66, wait 20ms, transmit 22 22 22, wait 0.5s, transmit 66 66 66, wait 20ms, transmit 22 22 22…
- `DISABLE usermsg4; DISABLE txgrp6;`
  //Disallow User Message #4 from transmission; stop transmitting transmit group #6; the Netway device will now only transmit 55 55 55, wait 1s, transmit 55 55 55… (assuming the `QUE txgrp3` command above was executed earlier in the same Emulation Program)

digoutN, freq(Hz)N, duty(%)N, and analogN

digoutN, where N is from 0 to 8: Used to reference one of the Netway device's 8 digital outputs. Depending on the command, digoutN operands can be used as either target or source operands, however if the output is not initialized before use as a source operand then the result will be unpredictable. Data from memory (Block Memory, X-, Y-, and Z-Variables) and received messages can be sent to a digoutN operand with the **SET** command, but because each digital output represents one bit of information, the only values valid for a digoutN operand are 0 and 1. If the value assigned to a digoutN operand is 0, then the corresponding digital output is said to be passive (effectively an open circuit with a closed driving transistor). Conversely if the value assigned to a digoutN operand is 1 (any nonzero value will be interpreted by the Emulation Program as 1), then the corresponding digital output is said to be active (effectively a short circuit to ground with an open driving transistor). A special reference to the digoutN operands, digout0, is included in the Emulation Programming Language for the purpose of setting the state of all 8 digital outputs with one Command Statement. The values of the 8 digital outputs are expressed a single byte (with digout8 as the most significant bit) when digout0 is used as an operand. For example: the Command Statement **SET** digout0 = 55h; will set digout1, digout3, digout5, and digout7 to 1 or active and set digout2, digout4, digout6, and digout8 to 0 or passive; this type of Command is usually found as an initialization at the beginning of Emulation Programs. digoutN operands can also be employed in conditional programming commands (**IF**/**ELSEIF**).
Examples: (assume x1 = $FF and a message begging with the byte 01 has been received)

- **SET** digout0 = 00h; **SET** digout7 = x1;
  //Set all the digital outputs (1-8) to passive; set digital output #7 to active
- **SET** digout4 = rxbyte0; **SET** x7 = digout0;
  //Set digital output #4 to active; set the value contained in x7 equal to $48
- **IF** digout0 & 80;
  //Statement true if only digout8 is active

pwmFreqN, where N is from 1 to 4 (only 2 & 4 valid for Netway6): Used to reference the frequency characteristic of the PWM Outputs. A Netway device has 4 PWM Outputs, and using pwmFreqN references PWM Output #N. Therefore pwmFreq1 references PWM Output #1, … , and pwmFreq4 references PWM Output #4. pwmFreqN operands can only be used as target operands, and are able to hold 16-bit values (numerical range: 0-65535 decimal, or frequency range: 0-65535 Hz). Data from Y- and Z-Variables can be sent to a pwmFreqN operand with the **SET** command.
Examples: (assume y1 = $0DDD and z6 = $AAABBAAA)

- **SET** pwmFreq1 = y1;
  //Set the frequency of PWM Output #1 equal to 221 Hz ($00DD)
- **SET** pwmFreq2 = z1;
  //Set the frequency of PWM Output #2 equal to 47786 Hz ($BAAA)
- **SET** pwmFreq3 = F0F0h;
  //Set the frequency of PWM Output #1 equal to 61680 Hz ($F0F0)

pwmDutyN, where N is from 1 to 4 (only 2 & 4 valid for Netway6): Used to reference the duty cycle characteristic of the PWM Outputs. A Netway device has 4 PWM Outputs, and using pwmDutyN references PWM Output #N. Therefore pwmDuty1 references PWM Output #1, … , and pwmDuty4 references PWM Output #4. pwmDutyN operands can only be used as target operands, and can be assigned 8-bit numerical values ranging from 0-100 decimal which correspond to duty cycle values ranging from 0-100% (an Emulation Program will set a duty cycle of 100% if a value greater that 100 decimal is assigned to a pwmDutyN operand). Data from X-Variables, analog inputs, and received messages can be sent to a pwmDutyN operand with the **SET** command.
Examples: (assume x1 = $FF, analog2 = 92, and the message 11 22 33 44 was just received (not on CAN1))

- **SET** pwmDuty1 = x1; **SET** pwmDuty2 = analog2;
  //Set the duty cycle of PWM Output #1 equal to 100%; set the duty cycle of PWM Output #2 equal to 92%
- **SET** pwmDuty3 = rxbyte3; **SET** pwmDuty4 = 39;

//Set the duty cycle of PWM Output #3 equal to 44%; set the duty cycle of PWM Output #4 equal to 39%

`analogN`, where `N` is from 0 to 12: Used to reference the analog inputs and outputs of a Netway device. Netway devices have 8 analog inputs and 12 analog outputs (provided by an optional daughterboard). Using `analogN` as a source operand signifies a reference to analog input #N, whereas using `analogN` as a target operand signifies a reference to analog output #N. For the analog inputs, `analogN` contains the 8-bit analog-to-digital conversion value of analog input #N. This analog-to-digital conversion occurs at a rate of 61millivolts per bit every 1 millisecond and covers a full range from 0-15.6VDC (data values range from 0-FF hexadecimal or 0-255 decimal). For the analog outputs (8-bit resolution), `analogN` contains the 8-bit digital-to-analog conversion value for analog output #N and can be driven by a full range of data values (from 0-FF hexadecimal or 0-255 decimal) from X-Variables or numerical constants. A special reference to the `analogN` operands, `analog0`, is included in the Emulation Programming Language for the purpose of setting all 12 analog outputs to one value with a minimum amount of Command Statements. For example: the Command Statement `SET analog0 = 00h;` will set all 12 analog outputs to 0 Volts; this type of Command is usually found as an initialization at the beginning of Emulation Programs. `analogN` operands can also be employed in conditional programming commands (`IF`/`ELSEIF`).
Examples: (assume `x86` = $25)

- `SET analog0 = FFh; SET analog6 = x86; SET x6 = analog2;`
  //Set all the analog outputs to the analog equivalent of $FF; set analog output #6 to the analog equivalent of $25; set `x6` equal to the digital equivalent of analog input #2
- `IF analog3 == x86; ELSEIF x86 == analog5;`
  //True if the value of analog output #3 is the analog equivalent of $25; true if the value of analog input #5 is the analog equivalent of $25

diginpN, ifreq(Hz)N, iduty(%)N, and debounce

diginpN, where N is from 0 to 8: Used to reference the Netway device's 8 digital/analog inputs. A Netway device has 8 digital/analog inputs, and using diginpN references digital/analog input #N. Therefore diginp1 references digital/analog input #1, …, and diginp8 references digital/analog input #8. diginpN operands can be used as target or source operands in conditional programming commands (**IF**/**ELSEIF**), but because each digital/analog input represents one bit of information, the only values valid for a diginpN operand are 0 and 1. Digital/analog input signals between 0.00 and 3.00VDC are interpreted as logical 0s (diginpN = 0), and digital/analog input signals greater than 3.00VDC are interpreted as logical 1s (diginpN = 1). The state of individual diginpN operands can be saved in X-Variables with the **SET** command, and a special reference to the diginpN operands, diginp0, is included in the Emulation Programming Language for the purpose of expressing the state of all 8 digital/analog inputs with one Command Statement. The values of the 8 digital/analog inputs are expressed a single byte (with diginp8 as the most significant bit) when diginp0 is used as an operand. For example: if diginp1, diginp3, diginp5, and diginp7 are 1 and diginp2, diginp4, diginp6, and diginp8 are 0, then the Command Statement **SET** x1 = digout0; will set x1 to 55h; this type of Command is usually found as an initialization at the beginning of Emulation Programs. The digital/analog inputs are sampled at least every 1 millisecond (more likely every 100 microseconds) to accommodate most switching operations, however user controlled digital de-bouncing is available to slow this rate to 30 milliseconds (see the debounce section below for more information). This sampling rate dictates that the detection of a digital/analog input signal transition is recognized every 1 millisecond (or 30 milliseconds if de-bouncing is enabled), and therefore digital/analog input signals must be held at new levels for at least 1 millisecond (or 30 milliseconds if de-bouncing is enabled) to ensure the transition is detected. This is an important factor to consider when using digital/analog input signal transitions as Command Line Triggers, and when using the diginp0 operand because this value is also subject to change every 1 millisecond (or 30 milliseconds if de-bouncing is enabled).

Examples: (assume diginp6 and diginp7 are 1 and the other digital/analog inputs are 0)

- `SET x77 = diginp0;`
  //Set the value contained in `x77` equal to $60
- `IF digout5 == digout2; ELSEIF digout0 & 80;`
  //Statement true only if the value of digital/analog input #5 is equal to value of digital/analog input #2; statement true only if digital/analog input #8 is 1 and all other digital/analog inputs are 0

`pwmFreqN`, where `N` is 1 or 2: Used to reference the frequency characteristic of the PWM inputs. A Netway device has 2 PWM Inputs, and using `pwmFreqN` references PWM Input #N. Therefore `pwmFreq1` contains the frequency of PWM Input #1 and `pwmFreq2` contains the frequency PWM Input #2. The value of the `pwmFreqN` operands is calculated every 1 millisecond. `pwmFreqN` operands can be used as target or source operands in conditional programming commands (**`IF`**/**`ELSEIF`**), and are able to hold 16-bit values (numerical range: 0-65535 decimal, or frequency range: 0-65535 Hz). Data from `pwmFreqN` operands can be saved in X-, Y-, or Z-Variables with the **`SET`** command.
Examples: (assume `x25` = $86)

- `IF pwmFreq1 == 25h; ELSEIF x25 == pwmFreq1;`
  //Both true if the frequency of PWM Input #1 is equal to $86
- `SET y6 = pwmFreq2;`
  //Set the value contained in `y6` equal to the frequency of PWM Input #2

`pwmDutyN`, where `N` is from 1 or 2: Used to reference the duty cycle characteristic of the PWM Inputs. A Netway device has 2 PWM Inputs, and using `pwmDutyN` references PWM Input #N. Therefore `pwmDuty1` contains the duty cycle of PWM Input #1 and `pwmDuty2` contains the duty cycle of PWM Input #2. `pwmDutyN` operands can be used as target or source operands in conditional programming commands (**`IF`**/**`ELSEIF`**), and can hold 8-bit numerical values ranging from 0-100 decimal which correspond to duty cycle values ranging from 0-100%. Data from `pwmDutyN` operands can be saved in X-Variables with the **`SET`** command.
Examples: (assume `x25` = 40)

- `IF pwmDuty1 == 40; ELSEIF x25 == pwmDuty1;`
  //Both true if the duty cycle of PWM Input #1 is equal to 40%
- `SET x6 = pwmDuty2;`
  //Set the value contained in `x6` equal to the duty cycle of PWM Input #2

`debounce`: Used with the **`ENABLE`**/**`DISABLE`** commands to control the analog-to-digital conversion time for a Netway device's analog/digital inputs. The default analog-to-digital conversion rate is at least 1 millisecond (more likely 100 microseconds). This rate is adequate for most switching situations, however built-in de-bouncing processing is available to limit the maximum change of a signal's value to 8 per 1 millisecond, which dictates an analog-to-digital conversion time of 32 milliseconds (max. signal value / max. change per ms = 256 / 8 = 32ms). The `debounce` operand allows users to select the standard (1ms) or de-bounced (32ms) conversion time.
Examples:

- `ENABLE debounce; DISABLE debounce;`
  //Turn on de-bouncing (conversion time = 32ms); turn off de-bouncing (conversion time < 1ms)

filterN, log, replayN, and emulation

`filterN`, where `N` is from 1 to 32: Used to reference the up to 32 message filters defined in the Filters Tab-Window. See the Filters Tab-Window section for more information on message filters. `filterN` operands are used with the **`ENABLE`**/**`DISABLE`** commands to turn filters on/off. When a filter definition is enabled, messages are checked to see if they match that filter definition, whereas when a filter definition is disabled, messages are not checked against that filter definition. When a message matches a filter definition, that message is qualified to be logged and/or trigger the execution of a Command Line. Using the `filterN` operands is equivalent to setting the value in the Ena/Dis Column of the Filters Tab-Window. Examples:

- **`ENABLE`** `filter2;` **`DISABLE`** `filter27;`
  //Turn on filter definition #2; turn off filter definition #27

`log`: Used to with the **`ENABLE`**/**`DISABLE`** commands to turn logging on/off. A Netway2, 3, 4, or 5 Box can buffer up to 7020 messages in its internal circular Log buffer, and the Netway6 series can buffer up to 32000 messages in its internal circular Log buffer. The `log` operand dynamically controls the status of this logging feature, and is equivalent to the Enable Log Checkbox in the Operating Mode Setup Window. When the log is disabled, no messages will be logged into a Netway device's internal circular log buffer (see the Retrieving Logged Messages, Netway Memory, and Operating Mode Button sections for more information). Examples:

- **`DISABLE`** `log;` **`ENABLE`** `log;`
  //Turn off logging; turn on logging

`replayN`, where `N` is from 1 to 32000: Used with the **`ENABLE`**/**`DISABLE`** commands to dynamically control a Netway device's Replay Feature. During a Replay, the messages in a Netway device's log buffer are transmitted back onto the same network from which they were recorded with the same inter-message intervals that occurred when they were originally received and logged. The `replayN` operand is used to start and stop a Replay of a Netway device's internal log, where `N` is the number of

logged items to be replayed. Note: When the `replayN` operand is used with the **`DISABLE`** command the value of `N` is inconsequential. Examples:

- **`ENABLE`** `replay100;` **`DISABLE`** `replay100;`
  //Start transmitting the first 100 items in a Netway device's internal log; stop transmitting items from a Netway device's internal log
- **`ENABLE`** `replay32000;` **`ENABLE`** `replay32000;` **`DISABLE`** `replay4;`
  //Start transmitting the all the items in a Netway device's internal log; restart transmitting the all the items in a Netway device's internal log; stop transmitting items from a Netway device's internal log

`emulation`: Used to terminate the execution of an Emulation Program. The `emulation` operand is used only with the **`DISABLE`** command to totally halt Emulation Programs, and is equivalent to un-checking the Run Emulation Checkbox located at the top of the Command Display Area in the Netway Application Program's main window. If the Netway device is operating in stand alone mode (i.e. not connected to a PC), users must recycle power to the device to restart an Emulation Program once it has been disabled. Example:

- **`DISABLE`** `emulation;`
  //Terminate execution of the Emulation Program

offset+N, offset-N, and Numerical Constants

`offset+N`, where `N` is from 1 to (128 – Index Number of current Command Line) (Max 127): Used with the **`CL`** command to specify an offset relative to Command Line index number. The **`CL`** command is used within a Command Line to execute a different Command Line as a subroutine. The `offset+N` operand allows users to reference subsequent Command Lines by offset rather than index number; the value of the offset is determined by the `N` specified: offset = Command Line Index Number + `N`. Examples: (assume the following Command Statements are found in Command Line #5)

- **`CL`** `offset+2;` **`CL`** `offset+58;` **`CL`** `offset+123;`

  //Execute Command Line #7 (5+2) as a subroutine; execute Command Line #63 (5+58) as a subroutine; execute Command Line #128 (5+123) as a subroutine

`offset-N`, where `N` is from 1 to (Index Number of current Command Line - 1) (Max 127): Used with the **`CL`** command to specify an offset relative to Command Line index number. The **`CL`** command is used within a Command Line to execute a different Command Line as a subroutine. The `offset-N` operand allows users to reference preceding Command Lines by offset rather than index number; the value of the offset is determined by the `N` specified: offset = Command Line Index Number - `N`. Examples: (assume the following Command Statements are found in Command Line #55)

- **`CL`** `offset-2;` **`CL`** `offset-25;` **`CL`** `offset-54;`

  //Execute Command Line #53 (55-2) as a subroutine; execute Command Line #30 (55-25) as a subroutine; execute Command Line #1 (55-54) as a subroutine

Numerical Constant: Used to define numerical constants for commands and operands. Numerical constants can be used as the target operand for the **`CL`** and **`NETMODE`** commands, and as the source operand for the `xN`, `yN`, `zN`, `timerN`, `blockN`, `digoutN`, `pwmFreqN`, `pwmDutyN`, `infrdataN`, `txgrpN`, `rxbyteN`, `analogN`, `pwmFreqN`, `pwmDutyN`, and `diginpN` operands. Numerical constants can be specified as either decimal or hexadecimal values: hexadecimal numbers are specified with the `h` suffix (`Nh`), whereas decimal numbers have no suffix (`N`). To determine the valid range of numerical constants for any of

the above commands/operands refer to the specific section devoted to that command/operand. Examples:

- `CL 4; CL 40h;`
  //Execute Command Line #4 as a sub routine; execute Command Line #64 (40 hexadecimal)
- `NETMODE 4;`
  //Reset Single Wire CAN bus to normal mode signal level

rxbyteN (Copying Data from Received Messages)

rxbyteN, where N is from 0 to 11: Used to reference the bytes of a received message. rxbyteN operands (size: 1 byte) can be used as either target or source operands, however rxbyteN operands can only be a target operand in conditional programming statements (**IF**/**ELSE**). For all protocols except CAN Hardware Channel #1, rxbyte0 refers to the first byte of a received message, rxbyte1 refers to the second byte of a received message, …, and rxbyte11 refers to the twelfth byte of a received message. For CAN Hardware Channel #1, rxbyte0 refers to the first byte of a received message, rxbyte1 refers to the second byte of a received message, rxbyte2 refers to nothing, rxbyte3 refers to the third byte of a received message, …, and rxbyte11 refers to the eleventh byte of a received message.

Examples: (assume xN = $NN)

- **IF** rxbyte3 == 80h;
  //Statement true if byte #4 of the received message is $80
- **ELSEIF** rxbyte9 > x8;
  //Statement true if byte #10 of the received message is greater than $88

An Emulation Program can copy up to 12 data bytes from received messages by using rxbyteN as the source operand for the **SET** Command. The rxbyteN operand can be employed only within Command Lines whose execution is triggered by the detection of a received message (i.e. the Command Line's trigger must be defined in the Network Message area in the **Command Line Triggers Window** (see the Editing Triggers section for more information)). Therefore only Command Lines triggered by netmsgN, filterN, canNfltN, rxNetmsgN, or rxFilterN (and not Timers, Digital Input Signals, or non-trigger type events) can use rxbyteN as the source operand in **SET** Command Statements.

The following examples and explanation illustrate how to use the rxbyteN operand, and assume that the reception of the message 333 44 55 66 77 88 99 AA BB (not on CAN1) has triggered the execution of the Command Lines:

Copying Data from Received Messages to…	Command Statement	Result
X-Variables	**SET** x1 = rxbyte0;	x1 ß 03, x2 ß 33, x3 ß 44, x4 ß 55, x5 ß 66, x6 ß 77, x7 ß 88, x8 ß 99, x9 ß AA, x10 ß BB
	SET x1 = rxbyte9;	x1 ß BB
Y-Variables	**SET** y45 = rxbyte0;	y45 ß 0333, y46 ß 4455, y47 ß 6677, y48 ß 8899, y50 ß AABB
	SET y45 = rxbyte5;	y45 ß 7788
Z-Variables	**SET** z3 = rxbyte0;	z3 ß 03334455, z4 ß 66778899, z5 ß 0000AABB
	SET z3 = rxbyte3;	z3 ß 55667788
Block Memory	**SET** block2 = 100; **SET** block3 = 8; **SET** block1 = rxbyte2;	0100 ß 44, 0101 ß 55, 0102 ß 66, 0103 ß 77, 0104 ß 88, 0105 ß 99,

		0106 ß AA, 0107 ß BB

Explanation: For the X-, Y-, and Z-Variables, specifying `rxbyte0` as the source operand will result in the entire received message being transferred into memory as ascending sequential variables of the type specified, whereas specifying `rxbyte1`-`rxbyte11` will only copy the specified block (starting from the position specified: 1-byte for X-Variables, 2-bytes for Y-Variables, or 4-bytes for Z-Variables) into the specified variable. For Y-Variables, the first byte copied (specified by `rxbyteN`) is stored as the most significant byte of the Y-Variable, and the last byte to be copied (`rxbyteN+1`) is stored as the least significant byte of the Y-Variable. For Z-Variables, the first byte copied (specified by `rxbyteN`) is stored as the most significant byte of the Z-Variable, the second most significant byte gets the byte from position `rxbyteN+1`, the third most significant byte gets the byte from position `rxbyteN+2`, and the last byte to be copied (`rxbyteN+3`) is stored as the least significant byte of the Z-Variable. For Block Memory, the copying of begins by storing the byte at `rxbyteN` in the location specified by `block2`, and for each remaining byte to be copied, both `rxbyteN` and the `block2` value are incremented by 1 until the end of the message or block (size is determined by `block3`) is reached (see the Block Memory section for a detailed explanation of Block Memory). If the end of a message is encountered and there is still available space in a Y- or Z-Variable then erroneous data will fill the remainder of the segment of memory.

Note: for standard CAN messages, the 3-digit hexadecimal ID ($HHH) should be though of as 2-bytes of information ($0H and $HH) where the first byte is `rxbyte0` and the second is `rxbyte1`.

deviceState

In most cases the operand is used for read operations to get various device states as shown below: SET x1 = deviceStateN; First operand can be either X,Y or Z variables.
For example the serial number and timestamp are 4-byte values, thus *Z-variable* is more appropriate to use when getting those values as it is shown below: SET z2 = deviceState8; //get current timestamp
User can use this operand for write operation to set time-out period when bus is idle to allow sleep mode like so: SET deviceState11 = 60;//60 second to enter power-down mode when bus is idle.

Device State number	Description	Example
1	Get Flex-Ray operating mode	SET x1 = deviceState1;
2	Firmware version number	SET y1 = deviceState2;
3	Firmware built number	SET x1 = deviceState3;
4	Flex-Ray firmware version number	SET y1 = deviceState4;
5	Flex-Ray firmware built number	SET y1 = deviceState5;
6	Get device serial number	SET z1 = deviceState6;
7	Get Emulation name	SET x1 = deviceState7;
8	Get Time stamp in ms	SET z1 = deviceState8;
9	Get CAN1 bus state	SET x1 = deviceState9;
10	Get CAN2 bus state	SET x1 = deviceState10;

11	Set power-down interval	SET deviceState11 = 300;//300 seconds
12	Get self-test results	SET x1 = deviceState12;
13	Get RX_CAN1 message rate per second	SET y4 = deviceState13;
14	Get RX_CAN2 message rate per second	SET y5 = deviceState14;
15	Get RX_UBP1 byte rate per second	SET y6 = deviceState15;
16	Get RX_UBP2 byte rate per second	SET y7 = deviceState16;
17	Get cold-start flag (1-cold, 0-warm)	SET y8 = deviceState17;
18	Get hardware type	SET x1 = deviceState18;
19	Get board revision	SET x1 = deviceState19;
20	Get SD card free space in %	SET x1 = deviceState20;
21	Get SD card status	1 if card is inserted, otherwise 0
22	Get SD card log file is open	1 if file is open, otherwise 0
23	Get number of log files on SD card	number of log files

SD card

NW92 hardware features SD card interface capable of handling Class4 and higher SD cards. 8GB card is supplied when this premium option is purchased with the tool, this feature allows to record binary logs.

Open/Rename/Close log file

To open temporary log file (tmp.log) use this command:

ENABLE sdLog; // open log file

Once opened, logging messages into the files is started.

User can rename the file by performing this command:

SET sdLog = xN; //rename log file, name of the file is starting at xN

Use this command stopping the logging and to close the log file:

DISABLE sdLog; // close log file

Maintenance and sorting log files:

Deleting log files is done based on sorted by time list of files, after sorting is done, cat.lg file is created.
Sorting and creating cat.lg is automatically performed in few cases:

1. On Power-up
2. After Format SD
3. After any Deletion operation (see below)
4. When current log file is closed (DISABLE sdLog;), the new record is added in the end of cat.lg and number of log files is incriminated.

Note: Although cat.lg is created after power-up, the SD-EndOp event is not generated. In all other cases: deletion or formatting SD new cat.lg is created and SD-EndOp trigger is generated!

With 3000 log files on SD it takes less than 10 seconds to create cat.lg with time sorted list of log items.
User can perform a variety of maintenance operations by using these commands:

1. Format SD: ***SET sdLog = 0;****//format SD and create cat.lg*
2. Sort log files: ***SET sdLog - 0;****//perform sorting of all log files (cat.lg) by time in ascending order*
3. Get number of log files: ***SET yN = deviceState23;*** *//yN will contain number of log files on SD*
4. Date distribution:
    - ***SET y2 = sdLog;****//Distribution dates to y2-y12 (days 0...>=9)*
    - ***SET x2 = sdLog;****//dostobution dates to x2-x12 (weeks 0...>=9)*
5. New log files: when log file is closed it is automatically added to cat.lg
6. Deleting log files:
    - ***SET sdLog - N;//Delete N oldest files***
    - ***SET sdLog - xN;//Delete xN oldest files***

- ***SET sdLog - yN;//Delete yN oldest files***
- ***SET sdLog = yN;//Delete all files older than specified in yN (in days)***
- ***SET sdLog = N;//Delete all oldest files until N% of SD is free***

Depending on number of log files and number of user specified files to delete, the above operations could take from few seconds to minutes For example, with 3000 files on SD, the operation SET sdLog - 20; takes 12 seconds to complete - this includes re-sorting after deletion.

Additional useful commands:

SET yN = deviceState24; //yN will contain number of deleted log files from SD during last delete operation

SET xN = deviceState20; //free space available on SD card in %

SD event triggers:

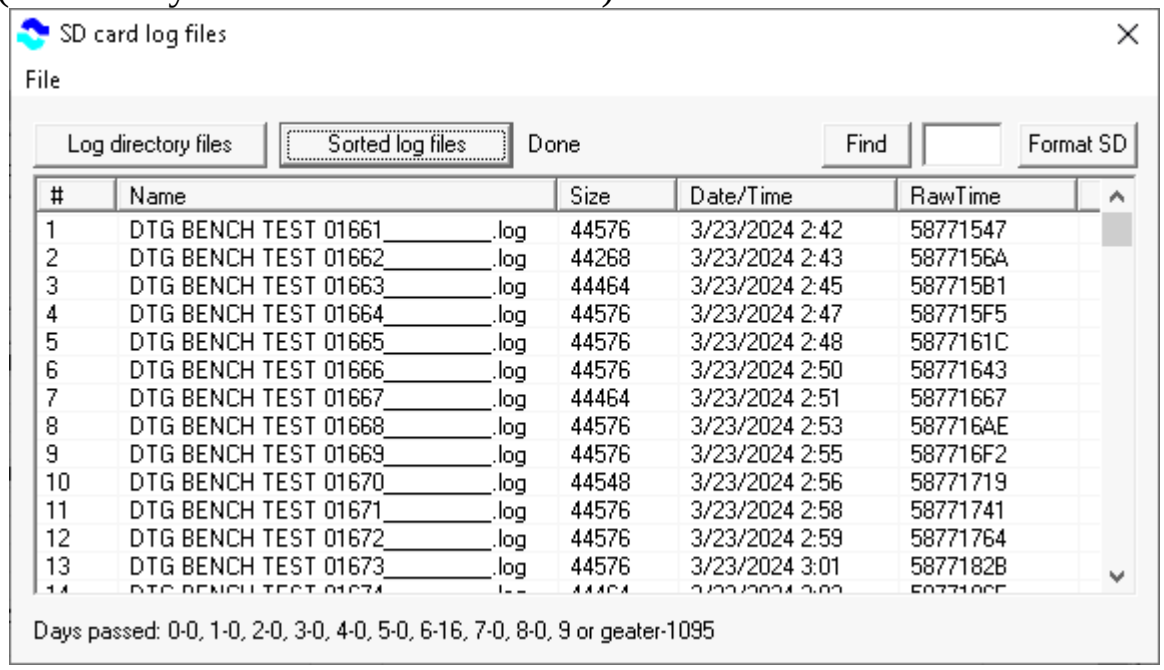
Due to lengthy SD operations during which tool appear to be non-operational, special event triggers are implemented to signal special SD events:

1. SD-Insert - SD card is re-inserted. Cat.lg file is NOT refreshed, To force refreshing use SET sdLog-0; triggered by this event.
2. SD-Eject - SD card is ejected. Recommended action: signal to the external tool that Netway is busy
3. SD-EndOp - End of lengthy operation, such as sorting or deleting files. Recommended action: signal to the external tool that Netway is ready to operate.

Before starting deletion of files, signal to the external tool that Netway is going to be busy.
Report readiness of the tool when SD-EndOp trigger happens.
During lengthy operation (deletion) and sorting process LED4 is lit red.

View SD card log files:

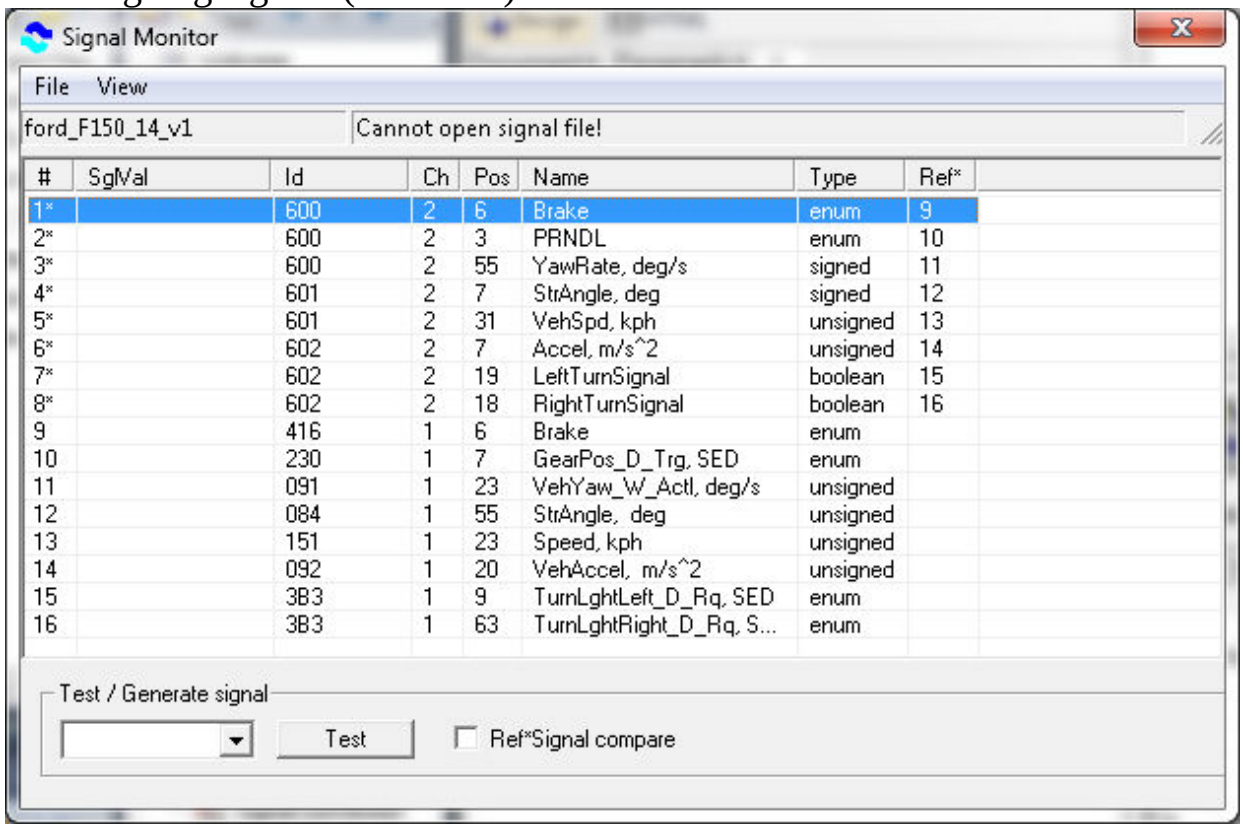
In Netway app use <Monitor/Explore SD> command to view this dialog (works only in USB connection mode):



The list of files after <Sorted log file> is displayed above.
Note: The status bar in the bottom of dialog is displaying Date Distribution of files. There are 16 files 6 days old and 1095 files older than 0 days on this SD.

Signals

This feature allows automatic conversion of bus signals. Typically there are some signals contained in incoming bus messages that need to be converted sometimes with different scaling and sent out to a different bus.
Example below shows how user can configure incoming signals (items 9-16) and outgoing signals (items 1-8):


Signal Monitor
File View
ford_F150_14_v1
Cannot open signal file!
SgVal Id Ch Pos Name Type Ref*
1* 600 2 6 Brake enum 9
2* 600 2 3 PRNDL enum 10
3* 600 2 55 YawRate, deg/s signed 11
4* 601 2 7 StrAngle, deg signed 12
5* 601 2 31 VehSpd, kph unsigned 13
6* 602 2 7 Accel, m/s^2 unsigned 14
7* 602 2 19 LeftTurnSignal boolean 15
8* 602 2 18 RightTurnSignal boolean 16
9 416 1 6 Brake enum
10 230 1 7 GearPos_D_Trg, SED enum
11 091 1 23 VehYaw_W_Actl, deg/s unsigned
12 084 1 55 StrAngle, deg unsigned
13 151 1 23 Speed, kph unsigned
14 092 1 20 VehAccel, m/s^2 unsigned
15 3B3 1 9 TurnLghtLeft_D_Rq, SED enum
16 3B3 1 63 TurnLghtRight_D_Rq, S... enum
Test / Generate signal
Test
Ref*Signal compare

Note that outgoing signals are configured as part of user messages on CAN2, while incoming signals are part of CAN filters on CAN1. Once configured and downloaded as part of emulation, conversion automatically takes place when incoming message is qualifies as CAN message filter, after calculation the signal value is taken into a specified user message according to the table above.
The tool allows to import signals from DBC file orJ1939/J1708 specification, or create the signal manually.

CommonNetway Type Library

Type topic text here.

NW92 44-pin Connector Assignment

Pin	Signal	Description
1	IO11	Analog I/ Digital IO / PWM -I / J1726-I
2	CAN1L	CAN1 Low
3	CAN1H	CAN1 High
4	CAN2L	CAN2 Low
5	CAN2L	CAN2 High
6	U1L	UART1 active low
7	U1H	UART1 active high
8	U2L	UART2 active low
9	U2H	UART2 active high
10	IO17	Analog / Digital IO
11	IO18	Analog / Digital IO
12	IO19	Analog / Digital IO
13	IO20	Analog / Digital IO
14	IO12	Analog I/ Digital IO / PWM -I / J1726-I
15	**GND**	Ground
16	IO1	Analog IO / Digital IO / PWM -O / J1726-O
17	IO2	Analog IO / Digital IO / PWM -O
18	IO3	Analog IO / Digital IO / PWM -O
19	IO4	Analog IO / Digital IO / PWM -O
20	IO5	Analog IO / Digital IO / PWM -O / J1726-O
21	IO6	Analog IO / Digital IO/ PWM -O / J1726-O
22	IO7	Analog IO / Digital IO/ PWM -O / J1726-O
23	IO8	Analog IO / Digital IO / PWM -O / J1726-O
24	IO9	Analog I/ Digital IO / PWM -I / J1726-I
25	IO10	Analog I/ Digital IO / PWM -I / J1726-I
26	IO13	Analog I/ Digital IO / J1726-I
27	IO14	Analog I/ Digital IO / J1726-I
28	IO15	Analog I/ Digital IO
29	IO16	Analog I/ Digital IO

30	CH3L	UART / CAN / FLEXRAY / SPI / I2C / J1850- / IO
31	CH3H	UART / CAN / FLEXRAY / SPI / I2C / J1850+ / IO
32	CH4L	UART / CAN / FLEXRAY / SPI / I2C / IO
33	CH4H	UART / CAN / FLEXRAY / SPI / I2C / IO
34	CH5L	UART / CAN / FLEXRAY / SPI / I2C / J1850-
35	CH5H	UART / CAN / FLEXRAY / SPI / I2C / J1850+
36	CH6L	UART / CAN / FLEXRAY / SPI / I2C / IO
37	CH6H	UART / CAN / FLEXRAY / SPI / I2C / IO
38	CH7L	UART / CAN / FLEXRAY / SPI / I2C / J1850- / IO
39	CH7H	UART / CAN / FLEXRAY / SPI / I2C / J1850+ / IO
40	CH8L	UART / CAN / FLEXRAY / SPI / I2C / IO
41	CH8H	UART / CAN / FLEXRAY / SPI / I2C / IO
42	REF 3V3	Reference 3.3V
43	REF 5V0	Reference 5.0V
44	**VBAT**	+12V Power source

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

NW80 FLEXRAY

Netway 80 Flexray 44-pin Connector Assignment:

Pin	Signal	Description
1	SLAVE_SEL	SPI/IIC Slave select output/input
2	CAN1_Low	CAN channel 1: HSCANL / FTCANL
3	CAN1_High	CAN channel 1: HSCANH / FTCANH
4	CAN2_Low	CAN channel 2: HSCANL
5	CAN2_High	CAN channel 2: HSCANH / SWCAN
6	U1L	UART channel 1: K-Line/RS485-
7	U1H	UART channel 1: RS485+
8	U2L	UART channel 2: K-Line/RS485-
9	U2H	UART channel 2: RS485+
10	U3L	UART channel 3: RS232RX
11	U3H	UART channel 3: RS232TX
12	J1850-	SCP- or SPI/IIC data output
13	J1850+	SCP+/Class2 or SPI Input
14	CLK_OUT	SPI / IIC Clock output/input
15	**GND**	**Ground**
16	IO1	Digital, PWM output/Analog, PWM input (0:12V)
17	IO2	Digital, PWM output/Analog, PWM input (0:12V)
18	IO3	Digital, PWM output/Analog, PWM input (0:12V)
19	IO4	Digital, PWM output/Analog, PWM input (0:12V)
20	IO5	Digital, PWM output/Analog, PWM input (0:12V)
21	IO6	Digital, PWM output/Analog, PWM input (0:12V)
22	IO7	Digital, PWM output/Analog, PWM input (0:12V)
23	IO8	Digital, PWM output/Analog, PWM input (0:12V)
24	IO9	Digital, PWM output/Analog, PWM input (0:12V)
25	IO10	Digital, PWM output/Analog, PWM input (0:12V)
26	IO11	Digital, PWM output/Analog, PWM input (0:12V) / Relay IN

27	IO12	Digital, PWM output/Analog, PWM input (0:12V) / Relay OUT
28	CAN4_Low	CAN channel 4: HSCANL
29	CAN4_High	CAN channel 4: HSCANH / SWCAN
30	CAN3_Low	CAN channel 3: HSCANL / FTCANL
31	CAN3_High	CAN channel 3: HSCANH / FTCANH
32	AIO1	Analog I/O (0:5V)
33	AIO2	Analog I/O (0:5V)
34	AIO3	Analog I/O (0:5V)
35	AIO4	Analog I/O (0:5V)
36	AIO5	Analog I/O (0:5V)
37	AIO6	Analog I/O (0:5V)
38	AIO7	Analog I/O (0:5V)
39	AIO8	Analog I/O (0:5V)
40	FlexrayA-	Flexray channel A BM
41	FlexrayA+	Flexray channel A BP
42	FlexrayB-	Flexray channel B BM
43	FlexrayB+	Flexray channel B BP
44	**VBAT**	VBAT +12V power

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

NW82

NW82 44-pin Connector Assignment:

Pin	Signal	Description
1	IO11/PWMIN	Analog/Digital/PWM input / Digital/PWM Output (0:5V)
2	CANL	CAN_Low channel 1
3	CAN1H	CAN_High channel 1
4	CAN2L	CAN_Low channel 2
5	CAN2H	CAN_High channel 2
6	U1L	UART channel 1 Low active
7	U1H	UART channel 1 High active
8	U2L	UART channel 2 Low active
9	U2H	UART channel 2 High active
10	SDA/IO12	I2C_SDA/Analog/Digital input / Digital/PWM Output (0:5V)
11	SCL/IO13	I2C_SCL/Analog/Digital input / Digital/PWM Output (0:5V)
12	J1850-/MOSI	J1850- / SPI Master-Out, Slave-In
13	J1850+/MISO	J1850+ / SPI Master-In, Slave-Out
14	CLK/IO14	SPI Clock / Analog/Digital input / Digital/PWM Output (0:5V)
15	**GND**	Ground
16	IO1	Analog/digital input (0:12V) / Digital/PWM output, 350mA,50V open drain
17	IO2	Analog/digital input (0:12V) / Digital/PWM output, 350mA,50V open drain
18	IO3	Analog/digital input (0:12V) / Digital/PWM output, 350mA,50V open drain
19	IO4	Analog/digital input (0:12V) / Digital/PWM output, 350mA,50V open drain

20	IO5	Analog/digital input (0:12V) / Digital/PWM output, 350mA,50V open drain
21	IO6	Analog/digital input (0:12V) / Digital/PWM output, 350mA,50V open drain
22	IO7	Analog/digital input (0:12V / Relay NO1 (0:60V, 2A)
23	IO8	Analog/digital input (0:12V / Relay NO2 (0:60V, 2A)
24	SRDY / IO9	SPI Slave ready / Analog/Digital input / Digital/PWM Output (0:5V)
25	SSEL / IO10	SPI Slave select / Analog/Digital input / Digital/PWM Output (0:5V)
26	U3L	UART channel 3 Low active
27	U3H	UART channel 3 High active
28	CAN10L/U10L	CAN_High channel 10 / UART channel 10 Low active
29	CAN10H/U10H	CAN_Low channel 10 / UART channel 10 High active
30	CAN3L	CANL_Low channel 3 / (UART channel 3 Low active - rev5)
31	CAN3H	CAN_High channel 3 / (UART channel 3 High active -rev5)
32	CAN4L / U4L	CAN_Low channel 4 / UART channel 4 Low active
33	CAN4H / U4H	CAN_High channel 4 / UART channel 4 High active
34	CAN5L / U5L	CAN_Low channel 5 / UART channel 5 Low active
35	CAN5H / U5H	CAN_High channel channel 5 / UART channel 5 High active
36	CAN6L / U6L	CAN_Low channel channel 6 / UART channel 6 Low active
37	CAN6H / U6H	CAN_High channel 6 / UART channel 6 High active

38	CAN7L / U7L	CAN_Low channel 7 / UART channel 7 Low active
39	CAN7H / U7H	CAN_High channel 7 / UART channel 7 High active
40	CAN8L / U8L	CAN_Low channel 8 / UART channel 8 Low active
41	CAN8H / U8H	CAN_High channel 8 / UART channel 8 High active
42	CAN9L / U9L	CAN_Low channel 9 / UART channel 9 Low active
43	CAN9H / U9H	CAN_High channel 9 / UART channel 9 High active
44	**VBAT**	VBAT +12V power

* - IO14/IO15 are connected to solid state relay (SSR) output pins, which are normal-open, with max ratings: 60V/2A
Digital output 14 controls SSR, so when:
SET digout14 = 1;, pins 10/11 are connected, SSR On
SET digout14 = 0;, pins 10/11 are disconnected, SSR Off

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

NW84

NW84 15-pin Connector Assignment:

Pin	Signal	Description
1	**VBAT**	VBAT +12V power
2	CAN1_Low	High-Speed CANL channel 1
3	CAN1_High	High-Speed CANH channel 1
4	CAN2_Low	High-Speed CANL channel 2
5	K-Line1	UART channel 1 for K-Line protocols
6	CAN2_High	High-Speed CANH channel 2
7	IO1	Analog/Digital input/output 1
8	J1850-	J1850- SCP
9	J1850+	J1850+ SCP/Class2
10	IO2	Analog/Digital input/output 2
11	n/c	not connected
12	RS485B	RS485B (J1708+ or ACP+)
13	RS485A	RS485A (J1708- or ACP-)
14	CAN2 SW	Single-Wire-CAN channel 2
15	**GND**	**Ground**

J1962 OBD-II - HDB15 adapter:
Netway J1962 adapter allows to plug NW84 directly into OBD-II connector, it is offered as one of the Netway accessories items:

J1962 Pin	HDB15 Pin	Signal	Description
1	1	CAN2 SW	Single-Wire-CAN channel 2
2	9	J1850+	J1850+ SCP/Class2
3	6	CAN2_High	High-Speed CANH channel 2
4	15	**GND**	**Ground**
5	15	**GND**	**Ground**
6	3	CAN1_High	High-Speed CANH channel 1
7	5	K-Line1	UART channel 1 for K-Line protocols
8	8	IO1	Analog/Digital input/output 1
9	9	IO2	Analog/Digital input/output 2
10	8	J1850-	J1850- SCP
11	4	CAN2_Low	High-Speed CANL channel 2
12	13	RS485A	RS485A (J1708- or ACP-)
13	12	RS485B	RS485B (J1708+ or ACP+)
14	2	CAN1_Low	High-Speed CANL channel 1
15	-		not connected
16	1	**VBAT**	VBAT +12V power

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

NW72

Netway 72 Series 44-pin Connector Assignment:

Pin	Signal	Description
1	IN1/IGN	Digital/analog input 1
2	CAN1_Low	CAN1 low
3	CAN1_High	CAN1 high
4	CAN2_Low	CAN2 low
5	CAN2_High	CAN2 high
6	K-Line1	K-Line 1
7	K-Line2/L-Line1	K-Line2
8	RS485_A1	RS485A ch1 / RS232_RX ch1
9	RS485_B1	RS485B ch1 / RS232_TX ch1
10	RS485_A2	RS485A ch2 / RS232_RX ch2
11	RS485_B2	RS485B ch2 / RS232_TX ch2
12	SCP-/ SPI_MRST	SCP-/ SPI_MRST
13	SCP+/SPI_MTSR	SCP+/SPI_MTSR
14	CLASS2	Class 2 (J1850+ GM/DCX bus)
15	**GND**	**Ground**
16	IN2	D/A input 2
17	IN3	D/A input 3
18	IN4	D/A input 4
19	IN5	D/A input 5
20	IN6	D/A input 6
21	IN7 / PWM_IN1	D/A input 7 / PWM input 1
22	IN8 / PWM_IN2	Digital/analog input 8 / PWM input 2
23	OUT1 / PWM_OUT1	OUT 1 / PWM output 1
24	OUT2 / PWM_OUT2	OUT 2 / PWM output 2
25	OUT3 / PWM_OUT3	OUT 3 / PWM output 3
26	OUT4 / PWM_OUT4	Digital output 4 / PWM output 4
27	OUT5	OUT 5

28	OUT6 / BUS15**	OUT 6 / BUS 15
29	OUT7 / BUS16**	OUT 7 / BUS 16
30	BUS1	BUS1
31	BUS2	BUS2
32	BUS3	BUS3 / OUT 8
33	BUS4	BUS4 / OUT9
34	BUS5	BUS5 / OUT 10
35	BUS6	BUS6 / OUT 11
36	BUS7	BUS7 / D/A input 9
37	BUS8	BUS8 / D/A input 10 / SPI_SS
38	BUS9	BUS9 / D/A input 11
39	BUS10	BUS10 / D/A input 12
40	BUS11	BUS11 /OUT 12
41	BUS12	BUS12 / IIC_SCL /OUT13
42	BUS13	BUS13 / IIC_SDA / OUT 14
43	BUS14	BUS14 / SPI_CLK /OUT 15
44	**VBAT**	VBAT +12V power
Abbreviations: D/A - digital/analog; OUT - open drain digital output;		

Netway 72 series with Daughter-Board Pin-out:

Pin	BUS	CAN daughter board	UART daughter board
30	BUS1	CAN3_Low	K-Line3
31	BUS2	CAN3_High	K-Line4
32	BUS3	CAN4_Low	K-Line5
33	BUS4	CAN4_High	K-Line6
34	BUS5	CAN5_Low	K-Line7
35	BUS6	CAN5_High	K-Line8
36	BUS7	CAN6_Low	K-Line9
37	BUS8	CAN6_High	K-Line10
38	BUS9	CAN7_Low	
39	BUS10	CAN7_High	
40	BUS11	CAN8_Low	
41	BUS12	CAN8_High	
42	BUS13	CAN9_Low	
43	BUS14	CAN9_High	
28	BUS15	CAN10_Low	
29	BUS16	CAN10_High	

Copyright © 2024-08-08 Smart Engineering Tools, Inc. All rights reserved.

ENDCL Command

The **ENDCL** Command is a no operand command used to terminate the execution of Command Statements within a Command Line. When **ENDCL** is encountered during the execution of a Command Line, no subsequent Command Statements in that Command Line will be executed.
The general form of the **ENDCL** Command is:

```
ENDCL;
```

The following example and explanation illustrates use of the **ENDCL** Command:

```
CommandStatementA1; CommandStatementA2; ...;
CommandStatementAN;
    ENDCL;
    CommandStatementB1; CommandStatementB2; ...;
CommandStatementBN;
```

//Command Statements A1-AN will be executed, however Command Statements B1-BN will not be executed

Note: All **CommandStatement**s referred to above can be any valid Command Statement contained in the Netway Emulation Programming Language.